800.92

Fritz Nestle
Eberhard Ostertag

BASIC
LOGO
PASCAL

NOVOTRADE

FRITZ NESTLE — EBERHARD OSTERTAG

BASIC, LOGO, PASCAL

NOVOTRADE

A könyv eredeti címe:
Kleiner Sprachführer BASIC, LOGO, Pascal (1986)

Fordította: MESKÓ KRISZTINA
RÁCZ MIHÁLY
Lektorálta: DR. LENGYEL JÓZSEF
PÁL LÁSZLÓNÉ

B 76079

A kiadásért felel Rényi Gábor, a NOVOTRADE Rt. igazgatója
Felelős szerkesztő: Siba László
Műszaki szerkesztő: Dévényi Erika
ISBN 963 02 5300 3
Hungarian translation © Meskó Krisztina, Rácz Mihály
Copyright © Markt & Technik Verlag Aktiengesellschaft
8013 Haar bei München (Germany)

TARTALOMJEGYZÉK

1. BEVEZETŐ

1.1 Mit várhatunk ettől a könyvtől? 7
1.2 Ébredés BASIC-kel, utazás LOGO-ország felé, érkezés Pascallal 8
  1.2.1 BASIC 8
  1.2.2 LOGO 8
  1.2.3 Hasonlóságok a BASIC és a LOGO között 9
  1.2.4 Pascal 9
  1.2.5 BASIC, LOGO és Pascal 10
1.3 Közvetlenül a képernyőre — példák a parancsokra 10

2. MEKKORA A DOBOZ?

2.1 A feladat meghatározása 12
2.2 Kidolgozás BASIC-ben 13
2.3 Megoldás LOGO-ban 16
2.4 Pascal programozás 20
2.5 A három nyelv különbségei 25

3. MI MARAD A PÉNZTÁRBAN?

3.1 A feladat meghatározása 28
3.2 Kidolgozás BASIC-ben 29
3.3 Megoldás LOGO-ban 31
3.4 Pascal programozás 36

4. KERESD AZ ÖSSZES OSZTÓT!

4.1 A feladat meghatározása 40
4.2 Kidolgozás BASIC-ben 41
4.3 Megoldás LOGO-ban 43
4.4 Pascal programozás 46

5. KAMATOS KAMAT SZÁMÍTÁSA

5.1 A feladat meghatározása 49
5.2 Kidolgozás BASIC-ben 50
5.3 Megoldás LOGO-ban 52
5.4 Pascal programozás 54

6. NÉHA SEGÍT EGY TÁBLÁZAT

6.1 A feladat meghatározása 57
6.2 Kidolgozás BASIC-ben 58
6.3 Megoldás LOGO-ban 61
6.4 Pascal programozás 64

7. HÉTOLDALÚ KOCKA?

7.1 A feladat meghatározása 68
7.2 Kidolgozás BASIC-ben 69
7.3 Megoldás LOGO-ban 72
7.4 Pascal programozás 75

8. DÉLOSZTÓL A JELENKORIG — KÖZELÍTŐ ÉRTÉKEK SZÁMÍTÁSA

8.1 A feladat meghatározása 79
8.2 Kidolgozás BASIC-ben 80
8.3 Megoldás LOGO-ban 82
8.4 Pascal programozás 85

9. BETŰTÖLCSÉR

9.1 A feladat meghatározása 87
9.2 Kidolgozás BASIC-ben 87
9.3 Megoldás LOGO-ban 89
9.4 Pascal programozás 91

10. RENDEZÉS

10.1 A feladat meghatározása 94
10.2 Kidolgozás BASIC-ben 95
10.3 Megoldás LOGO-ban 97
10.4 Pascal programozás 100

11. HOLNAP UGYANÚGY

11.1 A feladat meghatározása 104
11.2 BASIC programok betöltése, tárolása és kinyomtatása 104
11.3 LOGO programok betöltése, tárolása és kinyomtatása 106
11.4 Pascal programok betöltése, tárolása és kinyomtatása 108

12. PILLANTÁS A JEGYZETFÜZETBE

12.1 A feladat meghatározása 110
12.2 Kidolgozás BASIC-ben 113
12.3 Megoldás LOGO-ban 124
12.4 Pascal programozás 131

FÜGGELÉK

A. Iteráció és rekurzió a BASIC-ben, a LOGO-ban és a Pascalban 143
B. LOGO—LAYOUT 146

1. BEVEZETŐ

1.1 MIT VÁRHATUNK ETTŐL A KÖNYVTŐL?

Egy nyelvi kalauztól például nem szabad túl sokat várni. Az arra jó, hogy az ember ismeretlen környezetben is tudjon tájékozódni. Ehhez pedig a hétköznapi helyzetek adják a legjobb támogatást.
A programozásban az egyszerű feladatok jelentik a „hétköznapi helyzetet". Olyan feladatokat választottunk, amelyek tapasztalataink szerint hasznosak.
A feladatok többségét a könyvben tárgyalt mindhárom nyelven kétféle programmal oldjuk meg:

— alapprogram;
— felhasználói program.

Alapprogramon a legegyszerűbb algoritmusra redukált feladatot értjük. Ezek a programok a működésük ismerete nélkül nem használhatók. A felhasználó program a futás során megadja a felhasználó számára mindazokat az ismereteket, amelyek a program használatához szükségesek.

E könyvben elsősorban a programnyelvtől független, tipikus problémamegoldásokat akarjuk bemutatni. Ez azt jelenti, hogy a felhasználó szempontjait figyelembe véve oldjuk meg a feladatot, mintegy a képernyővel „beszélgetve". A felhasználónak szánt információkat minden program elején kiírjuk a képernyőre.

A különféle programozási nyelveknek, mint pl. a BASIC, igen gazdag irodalmuk van. Hiányoznak azonban az olyan jellegű munkák, amelyek arról tájékoztatnának, hogy hogyan írhatnánk át egy feladat megoldását az egyik programnyelvről a másikra. Reméljük, hogy az általunk választott egyszerű feladatokkal ehhez segítséget nyújthatunk.

Aki még sohasem dolgozott személyi számítógéppel, példáink alapján a három tárgyalt programozói nyelv bármelyikén betáplálhatja a feladatokat a számítógépbe, és a géppel meg is oldhatja azokat. Aki már szerzett tapasztalatokat e három nyelv valamelyikével, példáinkból megtudhatja, hogyan fordítható le a feladatmegoldás egy másik programnyelvre. Eközben persze, a felhasználó megismerkedhet a nyelv szerkezetével is.

Mind a kezdőknek, mind azoknak, akik már ismernek egy programnyelvet, szeretnénk sokat megtakarítani abból a munkából, amely egy programnyelv jobb megismeréséhez kell.

Itt kívánunk még néhány szót szólni a programok különféle géptípusokon való felhasználhatóságáról. A programokat úgy írtuk meg, hogy néhány utasítás kivételével — ezeket külön is megjelöltük — bármilyen számítógépen futtathatók legyenek, mindhárom nyelven.

A BASIC sok számítógépen már a bekapcsoláskor rendelkezésre áll, a másik két nyelv azonban nem. Rádadásul a Pascal-hoz és a LOGO-hoz nélkülözhetetlen egy lemezmeghajtó.

A CP/M vagy MS—DOS operációs rendszerű számítógépek mindhárom nyelven programozhatók. Az APPLE cég személyi számítógépei is kielégítik ezeket az igényeket.

Amennyiben más típusú számítógépe van, nézzen utána, hogy milyen programnyelvek kaphatók az adott géptípushoz.

A könyvben szereplő programokat a COMMODORE cég C 64-es és PC 20-as számítógépeire dolgoztuk ki úgy, hogy csekély módosítások mellett fussanak az IBM és ADDLE cég számítógépein is.

1.2 ÉBREDÉS BASIC-KEL, UTAZÁS LOGO-ORSZÁG FELÉ, ÉRKEZÉS PASCALLAL

1.2.1 BASIC

A BASIC nyelvvel azért foglalkozunk, mert a ma kapható legkisebb számítógépekbe is be van építve (ROM) és a bekapcsolás után azonnal ez a beépített BASIC jelentkezik. Nincs szükség arra, hogy lemezről vagy kazettáról töltsük be a tárba, tehát a számítógép BASIC-ben azonnal programozható.

A BASIC nyelv már közel 18 éve létezik, számtalan nyelvjárása fejlődött ki. Minden számítógépgyártó új BASIC parancsokat, utasításokat „talált fel", s ennek megfelelően az egyik típusú gépre írt program a legtöbbször nem fut másik géptípuson.

Szerencsére van egy minden számítógép által ismert BASIC alapszókincs. Arra törekedtünk, hogy a programokban csak ezek a szavak, utasítások szerepeljenek. Néhány kivételes esetben kénytelenek vagyunk a megfelelő számítógép kézikönyvére utalni. Általában azonban nem okozhat gondot egyetlen BASIC program „futtatása" sem.

1.2.2 LOGO

A LOGO nyelv amellett, hogy nagyon jól használható programnyelv, abban különbözik a BASIC-től, de a Pascaltól is, hogy külön parancskészlete van, amellyel a képernyőre rajzolhatunk. A kezdők, így játékos módon ismerkedhetnek meg a nyelvvel. A LOGO-val foglalkozó irodalom is bőven tárgyalja ezt a felhasználási területet. E könyvbe nem írtunk ilyen jellegű programokat.

A BASIC-kel ellentétben a LOGO rendszerint nincs előre beépítve a számítógép tárjába. A LOGO-t először — többnyire mágneslemezről — be kell tölteni a gépbe. Azt hogy ez a művelet hogyan történik, az adott számítógéptől függ, vagy a LOGO kézikönyvben vagy a számítógép gépkönyvében kell utánanéznünk.

Bár a LOGO majdnem egyidős a BASIC-kel, messze nincsenek olyan éles különbségek az egyes nyelvjárásai között, mint a BASIC-nek. Sajnos, amióta a LOGO mikroszámítógépeken is megtalálható, a helyzet kezd változni. Mindenesetre a LOGO-t maga a felhasználó is könnyen megváltoztathatja, úgy, hogy azért vissza lehessen állítani az eredeti állapotot. Különösen szembetűnő eltérés a LOGO és más nyelvek között az, hogy a LOGO mind angolul (angol parancsszavakkal és hibajelzésekkel), mind németül (német parancsszavakkal és hibajelzésekkel) rendelkezésre áll. A könyvben szereplő programok angol nyelvű változatát a COMMODORE C 64-esen fejlesztettük ki. Néhány apróságban, amelyre majd külön utalunk, a COMMODORE—LOGO különbözik az APPLE—LOGO-tól. Létezik már a LOGO nyelv magyar változata is. A kulcsszavak angol szavak, de a hibajelzések magyar nyelvűek.

1.2.3 HASONLÓSÁGOK A BASIC ÉS A LOGO KÖZÖTT

Akár BASIC-ben, akár LOGO-ban dolgozunk, a számítógép rögtön végrehajtja az utasításokat. Ez azt jelenti, hogy a legtöbb utasítást közvetlenül be lehet írni, és végre lehet hajtatni anélkül, hogy programot írnánk (számítógép mint számológép). Az ilyen nyelveket interpreternyelveknek (értelmezőnyelv) is nevezik. Lehetővé teszik, hogy egy parancs hatását kipróbálhassuk, és csak azután használjuk fel a programban. Az interpreternyelvek másik jellegzetessége, hogy a programkészítés ideje alatt, a már elkészült programrészletet bármikor el lehet indítani és így ellenőrizni. Ha az eredmény nem kielégítő, vagy ha a gép hibát jelez, a hibás sorokat azonnal ki lehet javítani.

1.2.4 PASCAL

A másik két nyelvvel ellentétben a Pascal fordítónyelv. A programot először, teljes egészében — egy szövegszerkesztő programmal (editor) — meg kell írni. Az így kapott ún. forrásprogramot át kell alakítani (fordítani) a számítógépnek érthető formába. Az átalakítást a Pascal fordító végzi el. A forrásprogramból létrehoz egy gépi kódú programot, amellyel a számítógép már tud dolgozni. Az előrehaladás a Pascal nyelvben nem olyan „spontán” mint a BASIC-ben vagy a LOGO-ban. A megoldásra váró feladatot már a program megírása előtt meg kell szerkesztenünk, különben túl sok hiba csúszik a programba.

A kész programot a fordító gépi kódú programmá alakítja át, és így kerül a gép tárjába.

A házi és a CP/M, ill. az MS—DOS operációs rendszerű személyi számítógépekre a legelterjedtebb Pascal fordítók, 1985 közepén, a következők voltak:

- ZOOM—Pascal és OXFORD—Pascal C 64-esre (a C 64-es nem DOS rendszerű);
- UCSD Pascal (Apple és IBM gépekre);
- Pascal MT+ (CP/M vagy MS—DOS);
- TURBO—Pascal (CP/M vagy MS—DOS).

A TURBO—Pascal fordító kényelmes editorjával és a fordítás nagyon nagy sebességével tűnik ki a többi közül. Programjainknál csak pár másodpercet vesz igénybe a fordítás, s a program futtatható. Mivel a Pascalban messze nincs olyan nyelvi zűrzavar, mint ami a BASIC-ben uralkodik, a könyvben tárgyalt programokat más fordítókkal is fel lehet dolgozni (kis korlátozás: a 12. fejezet adattár programja).

1.2.5 BASIC, LOGO ÉS PASCAL

A Pascal nyelv legfőbb jellemvonásaként azt hangsúlyoztuk, hogy fordítónyelv. Természetesen az is lehetséges, hogy egy kész BASIC programot utólag lefordítsunk, mert a BASIC-nek is léteznek fordítói. A C 64-esen a fordítók többsége nem gépi kódra, hanem ún. P-kódra fordít, s ezt egy interpreter dolgozza fel. A lefordított (kompilált) programok nagy előnye, hogy kb. 3—10-szer olyan gyorsan dolgoznak, mint az eredetiek. Hosszú feldolgozásoknál ez igen fontos szempont. A jövőben várható a LOGO fordító megjelenése is.

Sajnos a BASIC-ben messze nincs olyan kényszerítő erő a strukturált programozásra, mint a másik két nyelvben. Amit a LOGO és a Pascal már a nyelvi elemeivel kierőszakol, azt a BASIC-ben programozónak önfegyelemmel kell pótolnia. A BASIC-ben lehet „hirtelen felindulás"-ból programozni, és többnyire működőképes, futtatható programokat is kapunk (az ún. spagettikódban). Ez persze elősegíti az áttekinthetetlen programok készítését, pedig hosszú feladatok megoldásánál a BASIC-ben sem megyünk semmire kellő rendszeresség nélkül. Ezért a BASIC példáink is, a Pascal és a LOGO programozás analógiájára, rendszerezetten strukturáltan épülnek fel. Kezdettől fogva hozzá kell szoknunk a szerkesztő eljáráshoz. Enélkül nem lehet hosszú és áttekinthető programokat írni BASIC-ben sem.

1.3 KÖZVETLENÜL A KÉPERNYŐRE — PÉLDÁK A PARANCSOKRA

Mindhárom programnyelv valamilyen segédprogrammal éri el, hogy a számítógép a program szövegét kiírja a képernyőre, és ott meg is tudjuk változtatni. Ez mindhárom nyelvnél az editorral, a szerkesztőprogrammal végezhető el. Ennek a szerkesztőprogramnak a tulajdonságaitól függ, hogy mennyire könnyű a program beírása és javítása.

A tájékozódást a képernyőn a kurzor segíti. Az a jel, amely megmutatja a képernyőn a kiírás pillanatnyi helyzetét. A kurzor lehet villogó négyzet alakú mező vagy bármilyen más jel, számítógépenként különbözhet. A kurzormozgató billentyűket is különféleképpen jelzik, újabban általában nyíllal. A nyíl abba az irányba mutat, amerre a billentyű lenyomásakor elmozdul a kurzor. A C 64-es gépen a kurzormozgató billentyűkön kívül még az INS (insert) és DEL (delete) feliratú gombbal is mozgathatjuk a kurzort. Azonban, míg a kurzormozgató gombokkal anélkül mozoghatunk a képernyőn, hogy megváltoztatnánk a szöveget, a DEL gombbal törölni lehet, az INS gombbal pedig helyet csinálhatunk egy beszúráshoz. Azt, hogy az Ön gépének hogyan működik a billentyűzete és a szerkesztőprogramja (editorja), azt a számítógép gépkönyvéből tudhatja meg.

A BASIC és a LOGO programnyelvből a számítógépet számológépként is használhatjuk. Természetesen a beírásokat a sor végén mindig a RETURN (NEWLINE, CR stb.) gomb lenyomásával kell lezárni, hogy a számítógép tudomásul vegye a próbálkozásunkat. Ilyenkor olyan reakciókra is számíthatunk, amelyek első pillanatban meglepőek. Íme néhány példa:

Pr. nyelv	**Parancs**	**A gép reakciója**
BASIC:	PRINT FERI	0 (a FERI-t változóként értelmezi)
LOGO:	PRINT FERI	SAJNOS NINCS FERI NEVU ELJARAS
BASIC:	PRINT"FERI"	FERI
LOGO:	PRINT "FERI"	FERI"
BASIC:	3*4	(a gép látszólag nem csinál semmit, de ha a gépben program van, azt elrontja, mert programsorként értelmezi, és beírja a 3-as programsornak a *4-et)
LOGO:	3*4	=12
BASIC:	PRINT 3*4	12
LOGO:	PRINT 3*4	12
BASIC:	x=4:PRINT x	4
LOGO:	x=4 PRINT x	SAJNOS NINCS X NEVU ELJARAS
	MAKE "X 4 PRINT X	SAJNOS NINCS X NEVU ELJARAS
	MAKE "X 4 PRINT :X	4

Az utolsó példa azt mutatja, hogy a változókkal végzett műveletek a LOGO nyelvben bonyolultabbak, mint a BASIC-ben.

Hosszú elméleti fejtegetések helyett most azt tanácsoljuk, hogy válasszon ki egyet, a három programnyelv közül, és írja be ezen a nyelven az első programpéldát.

2. MEKKORA A DOBOZ?

2.1 A FELADAT MEGHATÁROZÁSA

A számítógéppel szeretnénk megállapítani annak a téglatest alakú doboznak a térfogatát, amelynek hosszúságát, szélességét és magasságát ismerjük. A kész program futtatásakor a képernyőre először a

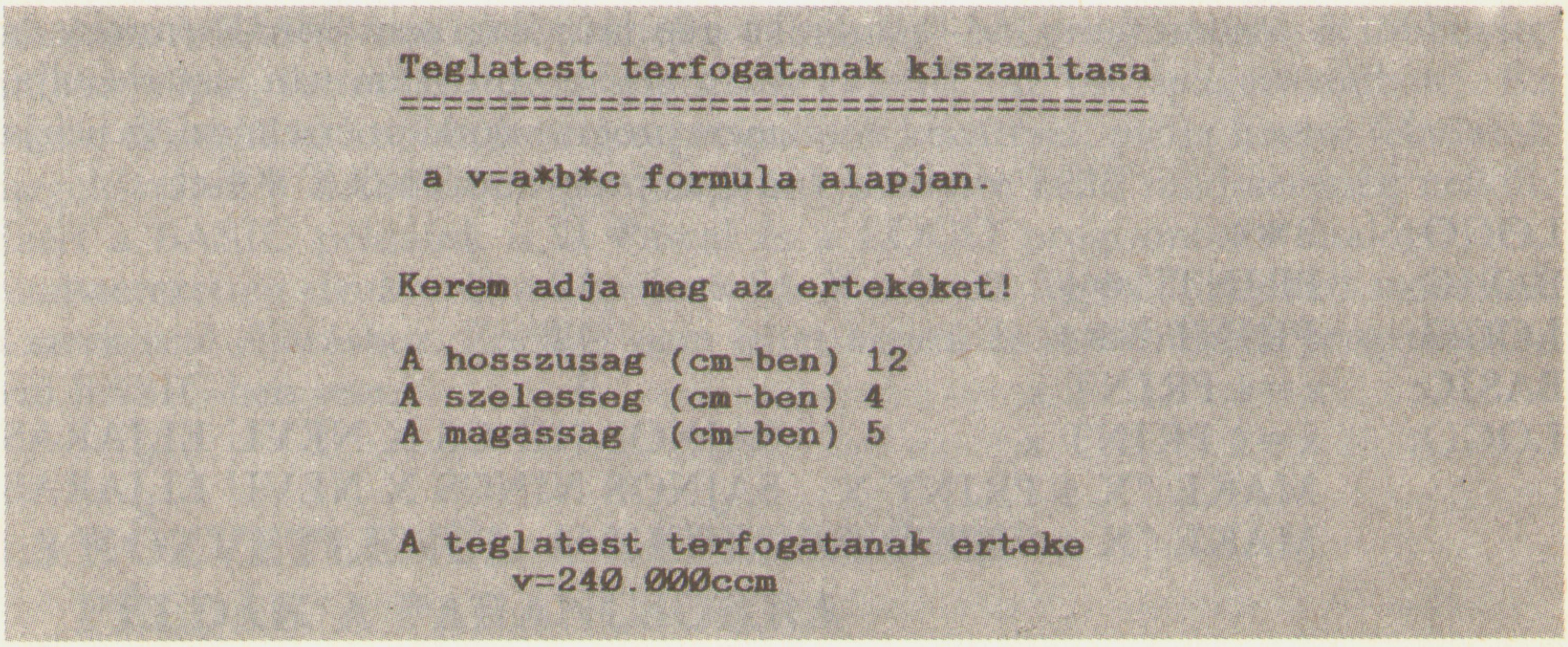
TEGLATEST TERFOGATANAK KISZAMITASA

feliratot íratjuk. Ezt a feliratot aláhúzzuk. Utána kihagyunk egy-két sort, és egy új sorban kiírjuk, hogy milyen képlettel akarunk számolni. Végül néhány üres sor után a

KEREM ADJA MEG AZ ADATOKAT:

felirat jelenik meg.
Ezt a részt szokás program fejlécnek nevezni.

Ezután az INPUT-rész következik. A számítógép kéri a hosszúság, a szélesség és a magasság adatait. Az adatok bevitele után a számítógép kiírja az eredményt a képernyőre. A program futtatása után a képernyőn kb. ez látszik:



```
Teglatest terfogatanak kiszamitasa
==================================

 a v=a*b*c formula alapjan.


Kerem adja meg az ertekeket!

A hosszusag (cm-ben) 12
A szelesseg (cm-ben) 4
A magassag  (cm-ben) 5


A teglatest terfogatanak erteke
     v=240.000ccm
```

A képernyőkiírást a BASIC programmal állítottuk elő.

A három nyelv apróbb eltéréseire mindenütt utalunk. Csak akkor tekintünk el ettől, ha az összehasonlítás aránytalanul nagy munkát igényelne. A különbségek közül most a **kész** jelre hívjuk fel a figyelmet, amellyel a gép a munka befejezését jelzi. A BASIC-ben a program futása a **READY.** felirattal fejeződik be és újra megjelenik a kurzor. A LOGO-ban a **kérdőjel** (?), vagy a magyar változatában a **nagyobb-jel** (>) és a kurzor látható ismét a képernyőn, ha lefutott a program. A jel (? vagy >) nem kerül feltétlenül a sor első helyére. Ha a LOGO program befejező utasítása a PRINT volt, akkor a jel az utolsó kiírt karakter után áll. Az előző feladat képernyőkiírásánál az eredmény „formájában" is lesz különbség. A Pascal mindig három tizedesjegyet ad meg, míg a BASIC és a LOGO elhagyja az eredmény végéről a nullákat.

2.2 KIDOLGOZÁS BASIC-BEN

Először az alapprogramot állítjuk össze. Itt is három részt különböztetünk meg: adatbevitel (input), számítás, kiírás (output). Ez az alapprogram öt sorból áll. Ezt az öt sort a BASIC-ben olyan sorrendben kell megszámozni, amilyenben az egyes parancsokat végre akarjuk hajtani. A sorszámok közötti különbség nagysága nem számít. Akkorára válasszuk, hogy adott esetben további sorokat is be tudjunk iktatni! Így a következő programot kapjuk:

```
190 INPUT A
200 INPUT B
210 INPUT C
310 V=A*B*C
630 PRINT V
640 END
```

A program beírásakor minden sor végén le kell nyomni a RETURN (NEWLINE, CARRIAGE RETURN, CR) gombot. Ha elkészültünk, indítsuk el a programot úgy, hogy beírjuk a RUN szót, és lenyomjuk a RETURN billentyűt. A gép először a 190-es sorban elhelyezett utasítást hajtja végre. Az INPUT A parancsnak a hatására egy kérdőjel jelenik meg a képernyőn, és a gép egy adat beírását várja tőlünk. Beírjuk a hosszúságot és lenyomjuk a RETURN gombot. A gép a beírt adatot az A változóba helyezi. Ezután megjelenik a következő kérdőjel (INPUT B hatására, 200-as sor), beírjuk a szélességét, ismét lenyomjuk a RETURN billentyűt. Végül a harmadik kérdőjel megjelenése után beírjuk a magasság adatát, és lenyomjuk a RETURN gombot. Most a beírt három szám az **A**, a **B** és a **C** változóban van. A 630-as sor (PRINT V) hatására a **V** tartalma kiíródik a képernyőre.

Ha egyszerű számokat adtunk meg, az eredményt fejben kiszámíthatjuk és összehasonlíthatjuk a gép által mutatott értékkel: ha a hosszúság, a szélesség és a magasság rendre 3, 4 és 5, akkor a képernyőn ez látható:

```
? 3
? 4
? 5
60
READY.
```

Ha az eredmény nem egyezik, valószínűleg hibát követtünk el a program beírásánál. A programlistát újra kiírathatjuk a képernyőre, ha begépeljük a LIST szót, majd lenyomjuk a RETURN gombot. Ekkor a számítógép növekvő sorszámok szerint kiírja a képernyőre a hat sort. Hasonlítsa össze jelről jelre, hogy az Ön által beírt sorok megegyeznek-e az itt szereplőkkel. A javításokat egyelőre úgy a legegyszerübb elvégezni, hogy a hibás sort a sorszámmal együtt újra beírjuk. Később, amikor már jobban ismeri számítógépe editorát (a számítógép kézikönyvében utána lehet nézni), lesznek egyszerübb javítási lehetőségek is.

Amennyiben a számítógéppel csak térfogatokat akar számíttatni, elég a most beírt program, hiszen tudja, hogy milyen adatokat vár a gép a három kérdőjelre. Ha azonban különböző programokat használ, hamarosan nehézségei lehetnek, mert nem kap a képernyőtől tájékoztatást, hogy éppen milyen program fut, illetve milyen adatot vár Öntől a gép.
Ezért a programot a következő programsorokkal egészítsük ki:

```
110 PRINT"□":REM CLS
120 PRINT"TEGLATEST TERFOGATANAK KISZAMITASA."
130 PRINT"==================================="
140 PRINT
150 PRINT"A V = A*B*C KEPLET SZERINT."
160 PRINT:PRINT
170 PRINT"KEREM ADJA MEG AZ ADATOKAT:"
180 PRINT
190 INPUT"HOSSZUSAG  A (CM-BEN) ";A
200 INPUT"SZELESSEG  B (CM-BEN) ";B
210 INPUT"MAGASSAG   C (CM-BEN) ";C

READY.
```

Általában egy sorba több utasítást is írhatunk. A legtöbb BASIC változat szerint ilyenkor kettősponttal kell elválasztani egymástól az utasításokat. Ha a képernyőn egy sorba csak 40 jel fér el, általában tovább írhatunk a következő sorban. Észrevehette, hogy a PRINT paranccsal nemcsak egy változó tartalmát, hanem szövegeket is kiírhatunk a képernyőre. Ilyenkor a kívánt szövegnek idézőjelben (” ”) kell állnia. Ugyanígy írhatunk ki az INPUT utasítással is szövegeket. A szöveg itt is idézőjelben áll. A változót, amelybe az adat kerül, pontosvessző (;) választja el a szövegtől. Javítsuk még ki ennek megfelelően a végeredmény kiíratását a következő három sor beiktatásával. A már meglévő 630-as sor ekkor automatikusan törlődik:

```
610 PRINT:PRINT
620 PRINT"A TEGLATEST TERFOGATA"
630 PRINT"      V = ";V;" CCM"
640 END

READY.
```

A LIST parancs begépelése után megjelenik a képernyőn a programnak egy terjedelmesebb változata. Kérjük, ellenőrizze még egyszer, hogy a képernyőn megjelenő programlista egyezik-e a könyvben lévővel. Ha így van, indítsa el a RUN paranccsal, és most sokkal „barátságosabb” képernyő feliratot fog látni.

A térfogatszámítás nagyon egyszerű példa. Mégis, már a legegyszerűbb feladatoknál is fűzzünk megjegyzéseket a programhoz. Szokjuk ezt meg, mert a későbbiekben meg fogja könnyíteni a dolgunkat hosszabb programok írásánál. A BASIC nyelvben a REM parancs vezeti be a kommentárokat. Amikor a program futtatása során a gép a REM szóba ütközik, figyelmen kívül hagyja az utána következő szöveget egészen a sor végéig.
A kiegészítés tehát:

```
10 REM TERFOGATSZAMITAS
20 REM ----------------------------------------------
30 REM A PROGRAM EGY TEGLATEST TERFOGATAT SZAMITJA KI
40 REM A HOSSZUSAGBOL, SZELESSEGBOL ES MAGASSAGBOL.
50 REM ----------------------------------------------
99 :

READY.
```

A további programrészeket is kommentáljuk. A 110-es sorban parancsot találunk a képernyő törlésére. Ez a parancs nem szabványosított, ezért minden BASIC változatban kicsit más. Saját kézikönyvében kell utánanéznie, hogy az Ön BASIC-jében mi az ennek megfelelő utasítás.

A megjegyzésben álló formája (CLS) az IBM-re érvényes. (APPLE: HOME, COMMODORE 64: PRINT CHR$(147))

```
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"□":REM CLS
299 :
300 REM A TERFOGAT KISZAMITASA
599 :
600 REM VEGEREDMENY KIIRASA ES A PROGRAM VEGE

READY.
```

Ha minden sort begépelt, nagyon „mutatós” felhasználói programot kapott. Mielőtt elindítaná, hasonlítsa össze még egyszer az általunk közölt programlistával.

```
10 REM TERFOGATSZAMITAS
20 REM ----------------------------------------------
30 REM A PROGRAM EGY TEGLATEST TERFOGATAT SZAMITJA KI
40 REM A HOSSZUSAGBOL, SZELESSEGBOL ES MAGASSAGBOL
50 REM ----------------------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"□":REM CLS
120 PRINT"TEGLATEST TERFOGATANAK KISZAMITASA"
130 PRINT"=================================="
140 PRINT
150 PRINT"A  V = A*B*C  KEPLET SZERINT"
160 PRINT:PRINT
```

```
170 PRINT"KEREM ADJA MEG AZ ADATOKAT:"
180 PRINT
190 INPUT"HOSSZUSAG  A (CM-BEN) ";A
200 INPUT"SZELESSEG  B (CM-BEN) ";B
210 INPUT"MAGASSAG   C (CM-BEN) ";C
299 :
300 REM A TERFOGAT KISZAMITASA
310 V=A*B*C
599 :
600 REM VEGEREDMENY KIIRASA ES A PROGRAM VEGE
610 PRINT:PRINT
620 PRINT"A TEGLATEST TERFOGATA:"
630 PRINT"     V = ";V;" CCM"
640 END

READY.
```

Új fogalmak a BASIC-ben:

INPUT, PRINT, PRINT; CLS (képernyőtörlés), END, REM és az értékadás. Ez utóbbi a LET V = A * B * C. Kényelmi okokból a legtöbb BASIC változatban a LET parancsszó elhagyható.

Ha e parancsok jelentése nem teljesen világos, nézzen utána egy BASIC tankönyvben.

2.3 MEGOLDÁS LOGO-BAN

Elsőként a LOGO-t kell betölteni a számítógép tárjába (l. 1. fejezet). Betöltés után a LOGO egy kérdőjellel jelentkezik be,
? vagy a magyar változat esetén >
amely azt jelzi, hogy parancs üzemmódban vagyunk. A kurzor a kérdőjel mögött áll és villog. A LOGO-ban a képernyő jelzi, hogy a gép parancs üzemmódban van-e, vagy éppen eljárásokat tanul, amelyeket később a parancs üzemmóddal hívhatunk elő. Ha programot akarunk írni, először át kell váltanunk a tanuló üzemmódra.
Most is az alapprogrammal kezdjük. Ahhoz, hogy tanulómódba jussunk, először írjuk be a ?-jel után az EDIT (vagy TO programnév) utasítást, majd nyomjuk le a RETURN gombot. Az EDIT parancs utasítja a LOGO-t, hogy kapcsoljon át a programeditorba. Az editor teszi lehetővé tetszőleges szövegek, tehát programok írását, javítását, bővítését (részeket törölhetünk, hozzáfűzhetünk vagy megváltoztathatunk). Az ehhez alkalmazott parancsok az adott számítógéptől függnek, de minden esetben kiírhatók a képernyő egy tetszőleges részére a kurzormozgató billentyűkkel.
Ha most begépelünk valamit, a szöveg megjelenik a képernyőn. Ha egy jelet törölni akarunk, vigyük a kurzort a törölni kívánt jelre és nyomjuk le egyszerre a CTRL és a D gombokat. Ekkor a jel törlődik, és a tőle jobbra álló szöveg egy

hellyel balra tolódik. (Kézikönyvében további lehetőségeket találhat egyes jelek vagy teljes sorok törlésére.)
Bármikor kiegészíthetjük a szöveget további jelekkel: vigyük a kurzort arra a helyre, ahova újabb jelet akarunk írni, és írjuk is be. Az utána következő szöveg automatikusan jobbra tolódik.

Aki a LOGO-val kezd dolgozni, eleinte különösnek találja a rendkívül sok helyköz (space) használatát. A LOGO a helyközt alkalmazza a két input elválasztására, és „érzékenyen" reagál, ha kifelejtjük.

Az, hogy az editorban, azaz a tanuló üzemmódban vagyunk, arról ismerhető fel, hogy a képernyő alsó sorában utalást találunk a parancs üzemmódba való visszatérésre a beírt program „megtanulásával", vagy anélkül

```
EDIT: CTRL—C TO DEFINE, CTRL—G :TO ABORT
```

Most tehát az editorban vagyunk, és beírhatjuk az alapprogramot. A BASIC-kel ellentétben minden LOGO program kap egy nevet, amelyet az angol változat esetén a TO szó előz meg. A programunkat nevezzük V-nek (volume = térfogat). A név után meg kell adnunk a programban használni kívánt változókat. Három változóra lesz szükségünk, amelyet **L**-lel, **B**-vel és **H**-val jelölünk. A változókat a LOGO-ban úgy jelöljük, hogy egy karaktersorozat elé kettőspontot írunk (:). Ha ezenkívül még azt is figyelembe vesszük, hogy minden parancsot, nevet és változónevet egy-egy szóközzel (space) el kell választanunk, begépelhetjük a program első sorát (a sor végén ne feledkezzünk meg a RETURN gomb lenyomásáról).

```
TO V :L :B :H
```

A további programsorok pedig, így alakulnak:

```
MAKE "V :L * :B * :H
```

A sor végén itt is le kell nyomni a RETURN gombot. A MAKE "V utasítás hatására a LOGO beírja a **V** nevű változóba az **:L * :B * :H** szorzat eredményét. Ne felejtsük el az idézőjelet a **V** előtt, különben a LOGO a jelet vagy a szót eljárásként értelmezi (l. 2.5 alfejezet). A BASIC-kel ellentétben a RETURN gomb lenyomásával nem kerül be a program a gép tárjába. Ehhez két billentyűt kell egyidejűleg lenyomni (CTRL C). Ez a képernyő alsó sorában is olvasható. A CTRL és C billentyűk lenyomására a gép ismét parancs üzemmódba kerül és a következő felirat jelenik meg a képernyő bal felső szélén:

```
PLEASE WAIT...
V DEFINED
?
```

Most már kipróbálhatjuk a programot, gépeljük be a következőket:

```
? V 3 4 5
```

és nyomjuk le a RETURN gombot. Ez a sor tartalmazza a program nevét és utasítja a LOGO-t az eljárás megindítására. A V-t követő három szám a hosszúság, szélesség és magasság értéke, ezeket az értékeket kapják az eljárás megfelelő változói.

A RETURN gomb lenyomása után látszólag semmi sem történik, a képernyőn az előző beírást követő sorban újra megjelenik a kérdőjel. Ha most beírjuk **:V** (lényeges a kettőspont a V előtt), akkor a

```
RESULT: 60
?
```

felirat jelenik meg a képernyőn.

Ez az alapprogram gyakorlatilag használhatatlan, mert tudnunk kell, hogy három számot kell beírni (hosszúság, szélesség, magasság) a futtatáshoz, hogy eredményt kapjunk. Ha kevesebb számot írunk be, a gép hibát jelez, az angol változatban:

```
V NEEDS MORE INPUTS
```

A programot a felhasználó szempontjai szerint alakítjuk ki ugyanúgy, mint a BASIC-ben. Négy részből építjük fel:

— programfejléc;
— adatbevitel;
— számítási rész;
— az eredmény megjelenítése.

Ezt a felépítést tekinthetjük egy eljárásnak is. Mielőtt beírnánk a gépbe, töröljük az előző programot (az angol változatnál ERASE V). Ezután az EDIT paranccsal menjünk át az editorba és írjuk be az alábbi programot:

```
TO TERFOGAT
 KEZDES
 ADATBEVITEL
 SZAMITAS
 KIIRAS
END
```

Ha ezt követően a CTRL és C gombokat lenyomva visszaváltunk parancs üzemmódba és elindítjuk a TERFOGAT programot, hibajelzést kapunk, miszerint a KEZDES nevű eljárás nincs definiálva. Ellentétben a MAKE vagy END szavakkal, ez a négy szó (KEZDES, ADATBEVITEL, SZAMITAS, KIIRAS) nem parancs vagy művelet, amit a LOGO nyelv ismer, hanem ezekhez előbb meg kell írni a megfelelő eljárásokat (részprogramokat) a LOGO parancsokkal és műveletekkel. Az EDIT-tel menjünk vissza a programeditorba és írjuk be a következő programot:

```
TO KEZDES
 NODRAW
 PRINT [TERFOGAT SZAMITAS]
 PRINT [=================]
 PRINT "'' PRINT [A V=A*B*C KEPLET SZERINT]
 REPEAT 2 [PRINT "’’]
 PRINT [KEREM ADJA MEG AZ ADATOKAT!]
 PRINT "’’
END

TO ADATBEVITEL
 PRINT [HOSSZUSAG A (CM-BEN)] PRINT "’ ’
 MAKE "A FIRST RQ
 PRINT [SZELESSEG B (CM-BEN)] PRINT "’ ’
 MAKE "B FIRST RQ
 PRINT "’ ’ PRINT [MAGASSAG C (CM-BEN)] PRINT "’ ’
 MAKE "C FIRST RQ
END

TO SZAMITAS
 MAKE "V :A * :B * :C
END

TO KIIRAS
 REPEAT 3 [PRINT "’’]
 PRINT [A TERFOGAT]
 PRINT "’     ’ PRINT1 [V=] PRINT1 :V PRINT1 "’  CCM ’ PRINT []
END
```

Az ND (NODRAW) és a CLEARTEXT utasítások az összes jelet eltűntetik a képernyőről és a kurzort a képernyő bal felső sarkába állítják. A PRINT utasítással szövegeket, vagy számítások eredményeit írhatjuk ki a képernyőre. A szövegeket szögletes zárójelbe írjuk. Ha egy változó értékét akarjuk kiírni, a változó neve elé kettőspontot kell tennünk. A PRINT utasítás hatására a kiírás mindig új sorban kezdődik, míg a PRINT1 utasításnál a kiírás a már megkezdett sorban folytatódik. A MAKE ” utasításban az idézőjel után álló jel változót jelent. Ha a fenti programot beírtuk, a CTRL és C gombok lenyomásával térjünk vissza a parancs üzemmódba. A gép a következő üzeneteket küldi:

```
PLEASE WAIT
Kezdés DEFINED
Adatbevitel DEFINED
Számítás DEFINED
Kiírás DEFINED
?
```

A program most már futtatható. Ehhez a kérdőjel után a TERFOGAT szót kell begépelni:

```
? TERFOGAT
```

A RETURN gomb lenyomása után elindul a TERFOGAT nevű eljárás az abban meghatározott sorrend szerint. Így kapjuk a már ismert képernyő feliratokat.

A program futtatása közben, a program által kért adatok beírása után mindig le kell nyomni a RETURN gombot. Az utolsó érték, a magasság beírását követően a gép kiírja az eredményt. Ugyanazt a képernyő felépítést kapjuk tehát, mint a BASIC program esetén.

Ha összehasonlítjuk a LOGO-beli megoldást a BASIC-belivel, megállapíthatjuk, hogy abból ami látszik a program felhasználója nem tudja megállapítani, hogy a programot milyen nyelven írták. Ez alól csak az indítás a kivétel, ti. A BASIC-ben RUN-nal indítunk, míg a LOGO-ban az eljárás vagy a program nevét kell beírni. Ha a LOGO-ban is RUN-nal akarunk indítani, akkor ezt az angol változat esetén megtehetjük úgy, hogy a RUN begépelése után a program nevét is beírjuk. Ellenkező esetben a "RUN NEEDS MORE INPUTS" hibajelzést kapjuk.

Ha a feladatot a TERFOGAT című főprogramnak megfelelően bontjuk részfeladatokra, megtakaríthatjuk a kommentárokat (a LOGO-ban), hiszen a részprogramok neveit a tartalmuknak megfelelően választhatjuk. Ha mégsem akarjuk kihagyni a kommentárokat, akkor ezeket pontosvesszővel válasszuk el a programnévtől, pl.:

```
TO KEZDES; Ez a reszprogram alakitja ki az ernyokepet
```

vagy, írhatjuk külön sorba is, de ekkor a pontosvesszőt (;) a sor elejére kell írnunk.

```
TO KEZDES
; Ez a reszprogram alakitja ki az ernyokepet
```

Új fogalmak a LOGO-ban:

ND, ill. CLEARTEXT, MAKE, PRINT, PRINT1, REPEAT, EDIT, END és az eljárások megírása.
A FIRST és RQ utasításokat a B. függelékben magyarázzuk meg. Az APPLE —LOGO-ban RQ helyett EG-t kell írni.

2.4 PASCAL PROGRAMOZÁS

Már az első fejezetben említettük, hogy a TURBO—Pascal fordítót fogjuk használni, és ezzel mutatjuk be, hogyan írhatunk programokat. A munka lényegesen nem változik más fordítók használatakor sem. Ha Pascal programot akarunk írni, mindenekelőtt egy editorra van szükségünk. A Pascal fordítók általában rendelkeznek editorral. A TURBO—Pascal indításához a számítógép bekapcsolása után először a diszkoperációs rendszert kell betölteni, majd a TURBO szó begépelésével a TURBO lemezről a programot. Ezután a fordító megkérdezi, hogy a hibaüzeneteket is betöltse-e. Akár Y(es)-t akár N(o)-t gépelünk be, a képernyőre kiíródik a „menü" (l. 11. fejezet). Most az E(ditor)-t választjuk, hogy a következő programot megírhassuk. Ahhoz, hogy az editorral kényelmesen tudjon dolgozni, jó, ha elolvassa a kézikönyv megfelelő fejezeteit.

Ha az OXFORD—Pascal áll rendelkezésére, akkor a LOAD"PASCAL",8(RETURN) paranccsal hívhatja be az úgynevezett „betöltő" programot, majd ez a RUN-nal elindítva betölti a Pascal többi részét is.

A BASIC-hez és LOGO-hoz hasonlóan először itt is az alapprogramot írjuk meg. A programot térfogatszámításnak nevezzük. A Pascalban minden program a PROGRAM szóval kezdődik, amelyet a programnév követ pontosvesszővel a végén. Így az alábbi kezdősorhoz jutunk:

```
PROGRAM TERFOGATSZAMITAS;
```

Ha OXFORD—Pascalt használunk, ne felejtsünk el sorszámot írni, ellenkező esetben „no source program” hibajelzést kapunk. Csak az első sorszámot kell beütni, utána automatikusan tízesével emelkedik a sorszám. Ezek után a programban előforduló változókat kell megneveznünk. Ezt hívjuk a változók deklarálásának. A deklaráció a VAR (változó) szóval kezdődik, majd felsoroljuk az első forduló összes változóit, egymástól vesszővel elválasztva. A felsorolás után kettőspontot írunk, és megadjuk a változók *típusát.* Ez a Pascalban négyféle lehet:

— integer (egész;)
— real (valós);
— char (karakteres);
— boolean (logikai).

A változó típusmegadása után pontosvesszőt kell írnunk.

Mivel a téglatest oldalai nem feltétlenül egész számok, valós típusú változókat kell használnuk. Így a változókat a következő módon deklaráljuk:

```
VAR h :REAL;
    sz :REAL;
    m :REAL;
     v:REAL;
```

vagy rövidebben:

```
VAR h,sz,m,v:REAL;
```

A deklarációs résszel elkészültünk. Rátérhetünk a feladat megoldására. Először beolvassuk a három adatot (readln (h,sz,m);), majd összeszorozzuk és az eredményt a V változóba tesszük (v: = h*sz*m), s végül kiíratjuk az eredményt a képernyőre [writeln (v);]. Ez a három utasítás alkotja a programot. A Pascalban ezt a részt a BEGIN és END. szavak közé kell helyezni. Ezzel a program az alábbi formát ölti:

```
program térfogatszámítás; var a,b,c,v:real;begin readln (h,sz,m);
v: = h*sz*m;writeln (v) end.
```

Ne ijedjen meg! Mivel a Pascal az egyes utasítások, deklarációk végét a pontosvesszőről felismeri, az összes utasítást közvetlenül egymás mellé írhatjuk. Természetesen sokkal áttekinthetőbb a program, ha az egyes utasításokat külön sorokba írjuk, és a program részeit üres sorokkal választjuk el, illetve bizonyos sorokat beljebb kezdünk. Így a következő formához jutunk:

```
program Teglatest_terfogata;
var h,sz,m,v : real;
begin
    readln(h,sz,m);
    v:=h*sz*m;
    writeln(v)
end.
```

(Figyelem! Az END után lényeges a pont!)

Elkészült a program, futtatni szeretnénk. Ehhez ki kell lépni az editorból (lenyomjuk a CTRL—K—D billentyűket), így visszajutunk a menühöz. Ott az R-t (RUN) választjuk és a számítógép gépi kódú programmá alakítja (kompilálja) a programunkat. Ez a folyamat néhány másodpercig tart. Ezután futni kezd a program. Írjuk be a téglatest méreteit úgy, hogy az egyes számokat üres helyközzel választjuk el egymástól. Az utolsó szám beírása után nyomjuk le a RETURN gombot. Az eredmény azonnal megjelenik a képernyőn:

```
3 4 5
60
>
```

Az OXFORD—Pascal használata esetén az L parancs indítja a fordítást, és az utána kiadott R parancs pedig a program futását, ha a program hibátlan. Ha csak az R parancsot adjuk ki, akkor is lefordítja a programot, de a lista nem jelenik meg a képernyőn.
Ha a program írásánál hibáztunk, a fordító azonnal észreveszi, és kiírja a megfelelő hibaüzenetet (hibajelet).
Ilyenkor a TURBO—Pascal az ESC gomb lenyomására visszatér editorba, mégpedig éppen arra a helyre, ahol a hibát elkövettük. A javítás után újra kezdődhet a fordítás.

Szóljunk néhány szót a Pascal nyelvű programok felépítéséről. Minden ilyen program az úgynevezett *programfej*jel kezdődik, és ezt követi a *programtörzs*. A programfej ad nevet a programnak. A PROGRAM szóval kezdődik, a program nevével folytatódik és pontosvesszővel zárul. Egyes fordítóknál a programnév és a pontosvessző közé egy listát kell írni, zárójelek közé, amely általában az input és output változókat tartalmazza. Egy teljes programfej így:

```
PROGRAM térfogat (input, output);
```

formájú.
A TURBO—Pascalban a programfej használata nem kötelező, és nincs is jelentősége a program szempontjából. Ennek ellenére leírjuk, de a részletes listát elhagyjuk. Ha gondjai vannak a Pascal programokkal, próbálja ki, hogy segít-e, ha megadja a listát.
A programtörzs ismét két részre oszlik:

— deklarációs rész;
— utasítás rész.

A deklarációs részben adjuk meg a programban szereplő összes álladót, a változókat és azok típusát, az eseleg felhasznált függvényeket és eljárásokat (részletesebben erről majd később szólunk, amikor az itt megemlített fogalmakat fel is használjuk a programokban). A példánkban csak változókat kellett deklarálni, a többit nem használjuk.

Az utasítás rész jelenti a tényleges programot. Itt találhatók a program végrehajtására vonatkozó Pascal parancsok a BEGIN és az END. között. (Ügyeljünk a pontra az END. után.)

Minden Pascal-deklarációt, illetve utasítást pontosvesszővel kell befejeznünk. A Pascal szavakat kis- és nagybetűkkel egyaránt írhatjuk.

Térjünk most vissza a programunkhoz. Ez az alapprogram sem ad futás közben semmiféle információt a felhasználónak. A BASIC-hez és LOGO-hoz hasonlóan itt is összeállítottunk egy olyan változatot, amelyből a felhasználó megtudhatja, hogy mire való a program, és milyen adatokat kell bevinnie a futtatás során.

```
program Terfogatszamitas;
var hossz,szelesseg,magassag:real;
                    terfogat:real;
begin
         clrscr;
         writeln('Teglatest terfogatanak kiszamitasa');
         writeln('==================================');
         writeln;
         writeln(' a v=a*b*c formula alapjan.');
         writeln;writeln;
         writeln('Kerem adja meg az ertekeket!');
         writeln;
         write('A hosszusag (cm-ben) ');readln(hossz);
         write('A szelesseg (cm-ben) ');readln(szelesseg);
         write('A magassag  (cm-ben) ');readln(magassag);
         terfogat:=hossz*szelesseg*magassag;
         writeln;writeln;
         writeln('A teglatest terfogatanak erteke');
         writeln('     v=',terfogat:5:3,'ccm')
end.
```

A programlistából látható, hogy kétféle módon írhatunk szövegeket a képernyőre:

— Writeln ('szoveg');
kiírja az aposztrófok, felsővesszők között álló szöveget a képernyőre. Ezután a kurzor a következő sor elejére ugrik, és a következő írási parancsra ott ír tovább.

— Write ('szoveg');
szintén kiírja a szöveget a képernyőre, de a kurzor nem ugrik át a következő sor elejére, hanem az utolsó leírt karakter mögött marad. A következő írási parancsra itt kezd tovább írni.

Ugyanez, érvényes az olvasási parancsra (readln) is, amellyel a billentyűzetről vihetünk be adatokat. A BASIC-kel ellentétben ezzel szöveg nem íratható ki. Az írási parancsok (write, writeln) — ugyanúgy, mint a BASIC PRINT parancsa

— szövegekre és a változók értékére egyaránt vonatkoznak. Arra kell csak ügyelnünk, hogy a szöveget a változótól válasszuk el egy vesszővel.

A Pascalban megtervezhetjük az eredményként kiírt számok formátumát. Ha, mint a példánkban is, az eredményt három tizedesjegyre kell kerekítenünk, először a változó nevét adjuk meg. Utána kettősponttal elválasztva két szám áll. Ezek közül az első annak a mezőnek a hosszúságát jelöli (a számjegyek száma, beleértve a tizedespontot is), amelybe a változó tartalmát írjuk balról jobbra haladva, a második pedig a tizedesjegyek számát adja meg. Nézzük meg mégegyszer az erre vonatkozó utasítást.

Writeln (' V=' , terfogat: 5: 3, ' ccm');

Ez az utasítás a térfogat értékét írja be balról jobbra haladva egy nyolc karakter szélességű mezőbe, három tizedesjegy pontossággal. Legyen például a térfogat értéke 12,3, akkor a képernyőn a

V = 12.300

felirat jelenik meg.
Ahhoz, hogy ennek a formátumozásnak a lehetőségeit minél jobban megismerje, kísérletezzen nyugodtan a WRITELN és a WRITE parancsokkal, változtassák meg a számokat is és figyeljék meg az eredményt a képernyőn.

A Pascalban is elhelyezhetünk kommentárokat, kerek zárójelbe, csillagok közé (* *) írva. A TURBO—Pascalban a kerek helyett kapcsos zárójeleket is használhatunk. Így kapjuk meg a Pascal program végleges változatát:

```
program Terfogatszamitas;
{ A program egy teglatest terfogatat szamolja ki }
{ a hosszusagbol,szelessegbol es a magassagbol.  }
var hossz,szelesseg,magassag:real;
                    terfogat:real;
begin
         clrscr;
         writeln('Teglatest terfogatanak kiszamitasa');
         writeln('==================================');
         writeln;
         writeln(' a v=a*b*c formula alapjan.');
         writeln;writeln;
         writeln('Kerem adja meg az ertekeket!');
         writeln;
         write('A hosszusag (cm-ben) ');readln(hossz);
         write('A szelesseg (cm-ben) ');readln(szelesseg);
         write('A magassag  (cm-ben) ');readln(magassag);
         terfogat:=hossz*szelesseg*magassag;
         writeln;writeln;
         writeln('A teglatest terfogatanak erteke');
         writeln('     v=',terfogat:5:3,'ccm')
end.
```

A képernyőt törlő CLRSCR utasítás csak a TURBO—Pascalra érvényes. Más fordító esetén az ahhoz tartozó kézikönyvben kell megkeresni. Az OXFORD—Pascalban például az erre vonatkozó utasítás a PAGE, de a WRITELN(chr(147)) is használható.

Új fogalmak a Pascalban:

PROGRAM, VAR, BEGIN, END, CLRSCR, WRITE, WRITELN, READ, READLN
adattípusok: INTEGER, REAL, CHAR, BOOLEAN, STRING.

2.5 A HÁROM NYELV KÜLÖNBSÉGEI

Foglaljuk most össze a három nyelv néhány tulajdonságát.

A legtöbb számítógépnél lehetőségünk van arra, hogy karakterkészletet válasszunk. A C 64-esnél például két készlet közül választhatunk:

1) kis- és nagybetűk (text mód);
2) nagybetűk és grafikus jelek (grafikus mód).

A SHIFT és a COMMODORE billentyűk együttes lenyomásával térhetünk át az egyik módról a másikra.
A BASIC-ben és a LOGO-ban nem keverhetjük a kis- és nagybetűket a kulcsszavakban, míg a Pascalban igen; a fordító mindkettőt megérti.
Természetesen mind a BASIC-ben, mind a LOGO-ban a PRINT utasítással vegyesen írhatunk ki nagy- és kisbetűket.
A BASIC-ben minden programsort sorszámmal kell kezdeni. A kész program futtatásakor a gép ebben a sorrendben hajtja végre az utasításokat. Az egyes sorszámok között érdemes elegendő helyet hagyni (általában 10 sorszámnyit), hogy később még további parancsokat is be lehessen iktatni. A másik két nyelvben nincsenek számozott sorok. A feldolgozás sorrendjét a programrészek előhívásának sorrendje határozza meg. A LOGO nagy előnye a másik két nyelvvel szemben a már említett kiterjeszthetősége. Ha olyan parancsra van szüksége, amelyet a LOGO nem ismer, írhat egy eljárást az elvégzésére. Ezt az eljárást tetszőleges névvel láthatja el. Ettől kezdve ezt az eljárást is ugyanúgy hívhatja elő, mintha beépített LOGO parancs lenne. Ugyanígy kezelünk minden, a gép tárában megtalálható eljárást. Ennek értelmében az eljárások nem mások, mint a LOGO parancskészletének kiterjesztései. Ezt a tulajdonságot használjuk fel, hogy kényelmesebb parancsokat hozzunk létre a képernyőre íráshoz. (L. még a 11. fejezetet és a B. függeléket!)
Mivel a programok, eljárások a számítógép kikapcsolásakor elvesznek, előzőleg érdemes őket mágneslemezre kiíratni. Kérjük olvassa el ehhez a 11. fejezetben leírtakat. Abból tudhatja meg, hogyan rögzítheti teljes egészében vagy részben a tár tartalmát (SAVE ”név). Így, ha később újra betölti a LOGO-t — ahogy azt a 11. fejezetben is leírjuk —, a lemezen tárolt programokat, eljárásokat visszatöltheti a tárba az újonnan megírt programok mellé és újra felhasználhatja azokat (READ ”név). A másik két nyelvben nem ilyen egyszerű a kiterjesztési lehetőség.
Mindhárom nyelvben vannak utasítások, amelyekkel a képernyőre írhatunk.

Mivel ezek a parancsok mind a felhasználónak adott utasítások, mind az eredmények ábrázolása szempontjából nagyon fontosak — szükség esetén a megfelelő kézikönyv segítségével —, meg kell ismerkednie velük. E tekintetben a BASIC, a LOGO és a Pascal hasonlóan viselkednek. A PRINT, a PRINT1 és a WRITE paranccsal írhatunk szövegeket vagy egyéb tártartalmakat a képernyőre.

BASIC

```
PRINT "szoveg1"; valtozo; "tovabbi szoveg"
```

LOGO

```
PRINT [szoveg1] PRINT1 :valtozo PRINT [tovabbi szoveg]
```

Pascal

```
Writeln ('szöveg1', valtozó, 'tovabbi szöveg');
```

Ha azt akarjuk pl. a BASIC-ben, hogy a következő PRINT utasítás ne kezdjen új sort, hanem az előzőt folytassa, akkor az előző PRINT utasítást pontosvesszővel kell lezárni. A LOGO-ban ezt a PRINT1 parancs eredményezi, míg a Pascalban erre a célre a WRITE utasítást használjuk.

Ha billentyűzetről viszünk be adatokat, akkor BASIC-ben az INPUT, a Pascalban a READLN és a READ utasításokat használjuk. A LOGO angol változatában erre a célra különféle parancsok használatosak. Az első példánkban a COMMODORE 64-esre írt LOGO REQUEST (RQ) parancsot használtuk. A LOGO-nyelv sajátosságai miatt, ha a billentyűzetről viszünk be adatokat, segédeljárást kell alkalmaznunk (l. B. függelék), amelyet INPUT-nak neveztünk el. Ezt az INPUT „parancsot” aztán, a BASIC INPUT, illetve a Pascal READLN és READ utasításának az analógiájára használhatjuk.

A BASIC-ben az INPUT paranccsal szöveget is ki lehet íratni a képernyőre az adatbevitel előtt. A LOGO-ban és a Pascalban az INPUT, illetve a READ és a READLN parancsok előtt kell külön utasítással a képernyőre írni. Ügyeljen továbbá arra is, hogy BASIC-ben az INPUT parancs után általában egy kérdőjel jelenik meg a képernyőn, és csak ezután lehet beírni a kívánt adatot.
Ha a Pascalban az adatbeviteli sorban az adatbevitel után tovább akarunk írni, akkor a READ parancsot kell használni a READLN helyett, ugyanez BASIC-ben is lehetséges. További különbség a három nyelv között a változók kezelésében mutatkozik. Míg a BASIC és a LOGO nem tesz éles különbséget az egyes számfajták (egész, valós) vagy egyéb kifejezések között, a Pascal — ahogy ez már korábban kiderült — a változók adattípusát is kéri.
A változók értékének megadása viszont az értékadó jel kivételével azonos (Y = 5 vagy LET Y = 5 a BASIC-ben, Y: = 5; a Pascalban).

A másik két nyelvtől teljesen eltérően, a LOGO minden szót először parancsként értelmez és megpróbálja végrehajtani, függetlenül attól, hogy a szó benne van-e a saját beépített szókészletében, vagy nincs. Ha például:

```
? PRINT
```

gépeljük be, a LOGO azonnal hibaüzenettel jelentkezik:

```
PRINT NEEDS MORE INPUTS (PRINT TÖBB ADATOT AKAR)
?
```

Akkor is ezt a hibajelzést adja, ha számára ismeretlen szó fordul elő. Azért, hogy a LOGO-nak egyértelmű legyen egy adatbevitelnél, hogy ez nem felszólítás, egy parancs végrehajtására, különböző jeleket, sajátos módon alkalmazunk. Rendkívül fontos a helyközjel (space), amelyet a LOGO parancsok, műveletek elválasztására használ. A másik jel a kettőspont. Ha név előtt áll, azt jelenti, hogy a név egy változó, amelynek értéket adhatunk és ezt az értéket használhatjuk. A LOGO-ban a változók bármilyen jelet tartalmazhatnak (betűt vagy számot), ellentétben a BASIC-kel vagy a Pascallal (l. még a 4.2 alfejezetet). A LOGO-ban a MAKE utasítással adunk értéket egy változónak.
Az utasítás formája: MAKE ”változónév
vagyis a parancsot egy helyköz követi, majd idézőjel, és ezután jön a változó neve kettőspont nélkül. A különleges jelek további sajátosságaira és jelentésükre példáink során még visszatérünk.

Az értékadó utsítások a következők:

BASIC

```
TERFOGAT = HOSSZ*SZELES*MAGAS
```

vagy

```
LET TERFOGAT = HOSSZ*SZELES*MAGAS
```

LOGO

```
MAKE ”TERFOGAT :HOSSZ * :SZELES * :MAGAS
```

Pascal

```
terfogat: = hossz*szeles*magas
```

Figyelje meg azt is, hogy a LOGO-ban és a Pascalban a változók nevei tetszőleges hosszúságúak lehetnek, míg sok BASIC értelmezőprogram (interpreter) ezt elfogadja ugyan, de csak a név első két karakterét veszi figyelembe. Ez azt jelenti, hogy a

HOSSZ és HOKI

változóneveket a LOGO és a Pascal különböző változóként kezeli, míg a BASIC értelmezők egy részének ez egy változót jelöl.

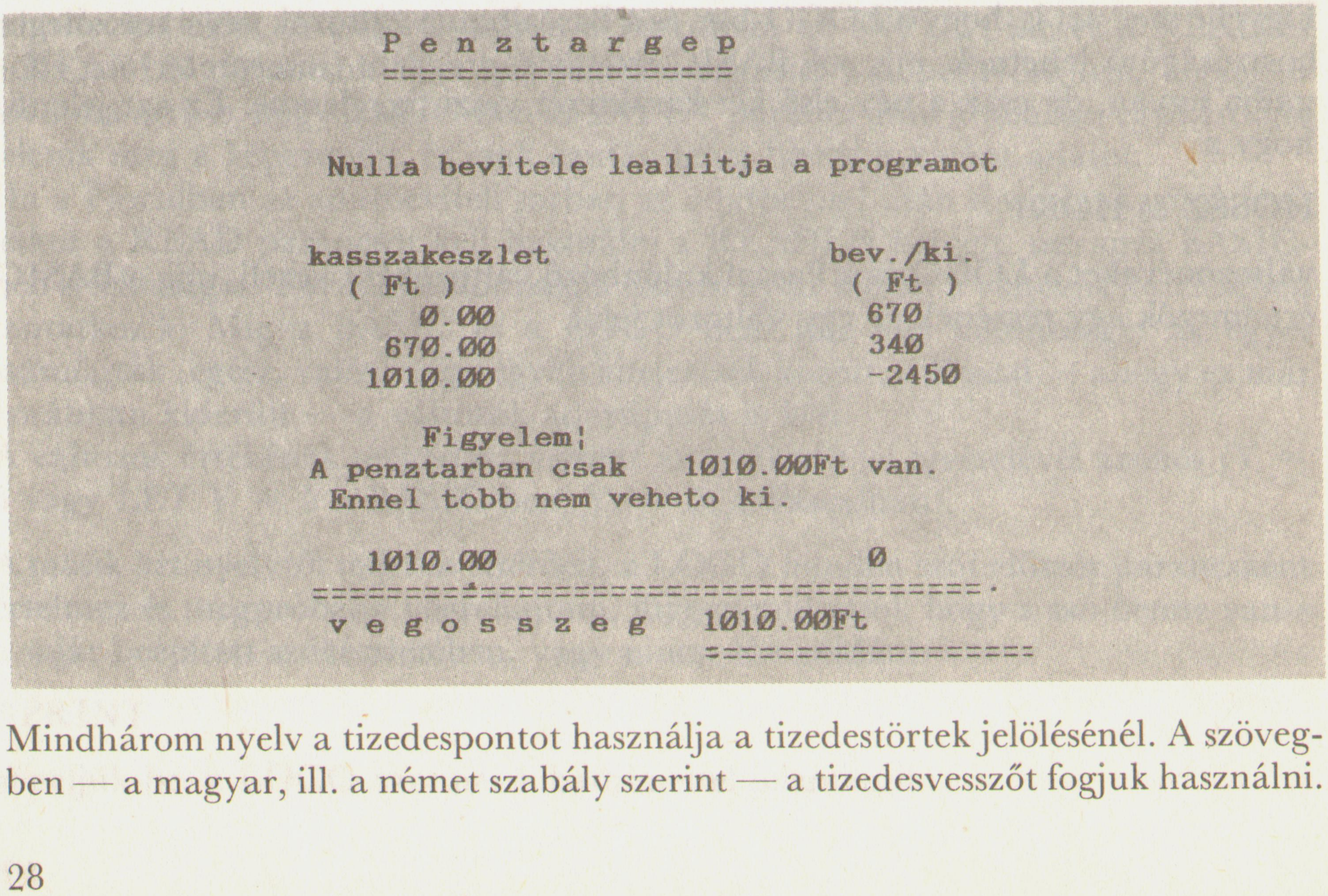
3. MI MARAD A PÉNZTÁRBAN?

3.1 A FELADAT MEGHATÁROZÁSA

Egy olyan programot készítünk, amellyel tetszőleges számú összeadandó összege kiszámítható. A program futása bármikor leállítható egy bizonyos szám beírásával (felfüggesztési feltétel), majd a végösszeg megjelenik a képernyőn.

Modellként egy úgynevezett egyenleg mutató pénztárgépet választunk. A képernyő bal oldalán jelenik meg a pillanatnyi egyenleg, a jobb oldalon pedig a folyamatos be- és kifizetések láthatók. A programfutás felfüggesztésére a nullát használjuk, ugyanis ilyen öszeg be- vagy kifizetésére a gyakorlatban nincs szükség. Valódi pénztárgép esetén legfeljebb a kasszában levő összegig lehet pénzt kifizetni. A programnak tehát vissza kell utasítania azokat a kifizetéseket, amelyek meghaladják a pillanatnyi egyenleget.

A felhasználó három különböző dolgot figyelhet meg a képernyőn:
1) a folyamatos be- és kifizetéseket, pozitív egyenlegénél;
2) a hibajelzést, abban az esetben, ha az egyenlegnél többet akarunk kifizetni;
3) a végösszeget.



```
    P e n z t a r g e p
    ==================

 Nulla bevitele leallitja a programot

kasszakeszlet               bev./ki.
  ( Ft )                     ( Ft )
      0.00                    670
    670.00                    340
   1010.00                    -2450

      Figyelem¦
A penztarban csak   1010.00Ft van.
 Ennel tobb nem veheto ki.

   1010.00                    0
====================================
 v e g o s s z e g   1010.00Ft
                     =================
```

Mindhárom nyelv a tizedespontot használja a tizedestörtek jelölésénél. A szövegben — a magyar, ill. a német szabály szerint — a tizedesvesszőt fogjuk használni.

3.2 KIDOLGOZÁS BASIC-BEN

Itt is, mint minden eddigi példánkban az alapprogrammal kezdjük, amelyet a program készítője vagy egy hozzáértő felhasználó ugyanolyan jól tud használni, mint egy tájékozatlan harmadik személy a részletesen kidolgozott programot.

A sorszámok közötti különbségeken semmi sem múlik, ezért ezeket úgy választjuk meg, hogy a részletes programban, kis kiegészítéssel, ismét jók legyenek:

```
READY.


350 PRINT SU
360 INPUT BE
370 IF BE=0 THEN 620
380 IF SU+BE>0 THEN 430
420 GOTO 350
430 SU=SU+BE
440 GOTO 350
620 PRINT SU

READY.
```

A BASIC programok elindításakor az összes felhasznált változó kezdeti értéke nulla, ezért **SU = 0.** A 350-es sor hatására a gép kiírja ezt az értéket, és továbbugrik a következő tabulátorpozícióra, mert a nyomtatási parancs vesszővel zárult. Ezután a 360-as sorban a gép egy szám beírására vár (INPUT BE). Az adat beírása után le kell nyomni a RETURN gombot, ezzel a **BE** változó aktuális értéke az imént beírt szám lesz. A 370-es sorban a programfutás felfüggesztésének a feltételét vizsgáljuk. Ha a **BE** változó értéke nulla, akkor a 620-as sorra ugrik a program, kiírja a képernyőre **SU** értékét, és a program futása befejeződik. Ha **BE** értéke nagyobb mint nulla, akkor a 380-as sor következik. Ott a program megvizsgálja, hogy az új egyenleg még nagyobb-e nullánál. Ha igen, akkor a 430-as sorban megállapítja az új egyenleget, majd a 440-es sornál visszatér a 350-es sorra és kezdődik a folyamat elölről. Ha azonban az egyenleg negatív lenne, akkor a 420-as sor téríti vissza a 350-es sorra úgy, hogy nem veszi figyelembe az előző adatbevitelt.

A program két logikai feltétele még a 430-as sorbeli összegzés elvégzése előtt található, a 370-es (felfüggesztés) és a 380-as sorban (nem lehet az egyenleg negatív). A programokat a 2.2 alfejezet szerint írjuk be, s a beírás után próbáljuk ki egyszerű számokkal. Így az eredményt fejszámolással is ellenőrizhetjük. Egy példa a felhasználó és a gép közötti párbeszédre:

```
0     ? 8
8     ? 7
15    ? −16
15    ? −12
3     ? 0
3
READY.
```

Ha a próbafuttatásnál más eredményre jutna, mint a számítógép, bizonyára a programsorok beírásánál követett el hibát. Ebben az esetben hívja elő újra a programot a LIST paranccsal és hasonlítsa össze jelről jelre az itt közölt programmal. Ha hibát fedez fel, akkor a 2.2 alfejezetben leírtak szerint kijavíthatja.

Az alapprogram az adott feladatot egyszerű módon oldja meg. Amennyiben a felhasználó tudja, hogy a gép mit vár tőle, és hogy mire való egyáltalán a program, akkor ebben a formában is használhatja. A következő kiegészítésekkel azonban sokkal „felhasználóközelibbé” alakítjuk a programot:

```
110 PRINT"♥":REM CLS
130 PRINT TAB(6);"P E N Z T A R G E P"
140 PRINT TAB(6);"==================="
150 PRINT:PRINT
160 PRINT"A '0' BEVETEL VAGY KIADAS BEFEJEZI A"
161 PRINT"PROGRAMOT!"
170 PRINT:PRINT
310 SU=0   :REM KEZDETI EGYENLEG
320 PRINT" EGYENLEG                BEVETEL/KIADAS"
330 PRINT"   (FT)                       (FT)     "
340 PRINT"---------------------------------------"
350 PRINT"  ";SU;TAB(29);
360 INPUT BE
370 IF BE=0 THEN 610
380 IF SU+BE>0 THEN 430 :REM EGYENLEG POZITIV
390 PRINT:PRINT TAB(16)"FIGYELEM!"
400 PRINT" A PENZTARBAN CSAK ";SU;"FT VAN."
410 PRINT" ENNEL TÔBB NEM FIZETHETO KI!"
420 PRINT:GOTO 350
430 SU=SU+BE
440 GOTO 350
610 PRINT"======================================="
620 PRINT"V E G O S S Z E G";TAB(27);SU;"FT"
630 PRINT TAB(27)"============="
640 END

READY.
```

Az alapprogramban lehetetlen volt többet kivenni a kasszából, mint amennyi benne volt, mert a **BE + SU < 0** esetben az egyenleg negatív lett (380-as sor). A program minden kommentár nélkül ismét az elejére ugrott vissza. Ebben a kiegészített programban a 390—420-as sorok figyelmeztetik a felhasználót a hibára.

A program most egy magyarázatos fejléc kiírásával kéri az adatok bevitelét, s megadja a kiszámított eredményt.

A programot RUN-nal indítva, és a megfelelő adatokat beírva a 3.1 alfejezetben bemutatott képernyőábrákat kell kapnia. Ellenkező esetben a program begépelésénél hibát követett el.

Azért, hogy a programot más is könnyen megérthesse, kommentárokkal egészítjük ki:

```
10 REM PENZTARGEP SZIMULACIOJA
20 REM ------------------------------------------------
30 REM A PROGRAM LEHETOVE TESZI TETSZOLEGES PENZOSSZEGEK
40 REM BEVETELET ES KIADASAT UGY, HOGY KIZARJA A NEGATIV
50 REM EGYENLEG LETREJOTTET.
60 REM ------------------------------------------------
99 :
100 REM PROGRAMFEJLEC
299 :
300 REM PENZTARGEP SZIMULACIOJA
380 IF SU+BE>0 THEN 430 : REM EGYENLEG POZITIV
599 :
600 REM EREDMENY ES VEGE

READY.
```

A REM sorok nem változtatják meg a program menetét. Mégis vizsgáljuk meg még egyszer, hogy a program helyesen dolgozik-e, mert figyelmetlenség miatt egy már kész sor megváltozhatott.

Új fogalmak a BASIC-ben:

feltételes elágazás IF—THEN, feltétel nélküli ugrás GOTO, értékadás kiterjesztése: SU = SU + BE (mivel az értékadásánál az egyenlőség nem szimmetrikus). Ezt a parancsot részletesebben LET SU = SU + BE formában is írhatjuk. PRINT TAB(n): a kiírás a sor n-edik helyén kezdődik.

3.3 MEGOLDÁS LOGO-BAN

Ismét abból indulunk ki, hogy az 1. fejezetben leírtaknak megfelelően a LOGO nyelvet betöltötte a számítógép tárába, és a B. függelék szerinti TAB, CR és INPUT eljárások is a rendelkezésére állnak. Újra a parancs üzemmódban vagyunk, ami a képernyőn megjelenő jelről ismerhető fel:

? vagy >

A térfogatszámításnál a LOGO közvetlen parancsait alkalmazhattuk a számításhoz alapprogramként. Most az alapprogram felépítése, a folyamatos adatbevitel miatt, jóval bonyolultabb. A programot három részre osztjuk:

— kezdés zérus egyenleggel;
— adatbevitel;
— ellenőrzés.

Az első részprogramot KASSZA-nak nevezzük el, és átkapcsolva a szerkesztőbe (EDIT KASSZA (RETURN)) rögtön meg is írjuk:

```
TO KASSZA
 MAKE "SU 0
 ADATBE
END
```

Ezzel, a kezdeti egyenleget állítottuk be nullára (MAKE ”SU 0). Ezután, meghívjuk a program második részét, az ADATBE eljárást:

```
TO ADATBE
 MAKE "BE FIRST RQ
 TEST :BE = 0
 IFTRUE PRINT ( :SU + :BE ) TOPLEVEL
 IFFALSE KONTROLL ADATBE
END
```

A program első sora azt jelenti, hogy a gép a billentyűzetről vár adatot és a **:BE** változó ezt az értéket kapja. A második sorban azt vizsgáljuk (TEST :BE = 0), hogy a bevitt adat nulla-e vagy sem (a program leállításának a feltétele). Ha nulla (IFTRUE), vagyis teljesül a feltétel, akkor a gép kiírja a pillanatnyi egyenleget és befejeződik a program futása (TOPLEVEL). Ha a feltétel hamis (IFFALSE), meg kell nézni, hogy az esetleges kifizetés (BE értéke negatív) túllépi-e a kassza tartalmát. Ehhez hívjuk elő a KONTROLL eljárást! Az ellenőrzés után ismét az ADATBE programot hívjuk, hogy a következő adatot is bevihessük. Az ADATBE eljárás valamennyi utasítása — a KONTROLL kivételével — a LOGO *beépített* utasítása, így csak a KONTROLL eljárást kell megírnunk.

```
TO KONTROLL
 TEST :SU + :BE > 0
 IFTRUE MAKE "SU :SU + :BE PRINT :SU ADATBE
 IFFALSE ADATBE
END
```

A program tehát ellenőrzi, hogy az új egyenleg még pozitív-e (TEST :SU + :BE >0). Ha ez igaz (IFTRUE), akkor a gép kiszámítja az új egyenleget (MAKE ”SU :SU + :BE), s kiírja képernyőre (PRINT :SU), majd újabb adatbevitelre vár (ADATBE).
Ha a kifizetés esetleg túllépné a kassza tartalmát, a bevitt adatot a gép nem veszi figyelembe (nincs összegzés) és visszatér újra az adatbevitelhez (ADATBE). Ha mindent beírt, nyomja le egyszerre a CTRL és a C billentyűket. Ezzel térhet vissza a parancs üzemmódba, és a képernyőn az alábbiak jelennek meg:

```
PLEASE WAIT...
KASSZA DEFINED
ADATBE DEFINED
KONTROLL DEFINED
?
```

A KASSZA szó begépelésével indítható a program. A kurzor egy üres sorban villog és a gép várja az első adat bevitelét. Ha például beírjuk a 8-ast (RETURN), akkor kétszer jelenik meg a 8, először mint bevitt adat, másodszor mint az új egyenleg. Ezután írjunk be pl. 7-et, akkor ugyanígy, a 7 mellett az új egyenleg, a 15 is megjelenik a képernyőn. Ha most −16-ot írunk be, akkor a gép nem számolja ki az új egyenleget, hanem új adatra vár. Ha −12-t ír be, akkor 3-at kap egyenlegként. Zérus beírásával befejezheti a program futását. Így egyenlegként 3-at kapunk, és a gép ismét parancs üzemmódba kerül. Az előbb ismertetett „párbeszéd” a képernyőn így látszik:

```
?   8
8
?   7
15
?  -16
?  -12
3
?   0
3
```

A kért adatokat úgy választottuk meg, hogy a program elágazásait be tudjuk mutatni.

A most megírt programból még hiányzik a megfelelő tájékoztatás a felhasználónak. A képernyőn megjelenő folyamat inkább csak zavart okoz. Olyan képernyőre lenne szükség, mint amit a 3.1 alfejezetben mutattunk be. Mielőtt ezt a felhasználónak sokkal kedvezőbb programot megírnánk, töröljük a tárból az eddigi programjainkat.

```
? ERASE KASSZA
? ERASE ADATBE
? ERASE KONTROLL
```

Ezután csak a TAB, CR, ... (l. a B. függelék) segédprogramok lehetnek a tárban. Erről győződjünk is meg:

```
? PRINTOUT TITLES
```

Ezután a számítógép felsorolja a pillanatnyilag rendelkezésére álló össze eljárást:

```
TO INPUT
TO TAB :N :T
TO CR :N
TO S :N
TO U :N :Z
?
```

Most már megírhatjuk a kibővített, PÉNZTÁRGÉP című programunkat. A feladatot négy eljárásra bontjuk:

```
TO PENZTARGEP
 KEZDES
 ADATBEVITEL
 EGYENLEG
 HIBAJELZES
END
```

A következő részprogramok (eljárások) — csekély eltéréssel — ugyanazt a programfolyamatot valósítják meg a képernyőn, mint a BASIC programnál. A PENZTARGEP program először a KEZDES nevű eljárást hívja meg:

```
TO KEZDES
 ND MAKE "K 0
 TAB 6 [P E N Z T A R G E P] CR 1
 TAB 6 "'' U 29 "= CR 3
 TAB 4 [NULLA BEVITELE LEALLITJA A PROGRAMOT] CR 3
 TAB 2 "KASSZAKESZLET TAB 31 "BEV./KI. CR 1
 TAB 5 [[FT]] TAB 32 [[FT]] CR 1
 U 38 "- CR 1
END
```

A program első parancsa törli a képernyőt és a kurzort a bal felső sarokba viszi. A második parancs hatására, a sor hatodik helyéről indulva a PENZTARGEP felirat jelenik meg a képernyőn. A harmadik sor első utasítása húzza alá a címet, míg a második, a CR 3 utasítás ebben az esetben csak két üres sort hoz létre, mert a TAB utasításban egy PRINT1 parancs is rejlik. A PRINT1 parancsnál ugyanis a kurzor az eredeti sorban marad a kiírás után, így az első sorváltást itt használja fel a gép. A negyedik parancssor tartalmazza azt a feliratot, hogy nulla beírására leáll a program. Az itt levő CR 3 utasítás is csak két üres sort eredményez, hiszen itt is volt egy TAB utasítás.

A következő két sor különféle szövegei a szögletes zárójel LOGO-beli tulajdonságaira hívják fel a figyelmet. A szögletes zárójelek között álló listát íratjuk ki a PRINT utasítással, vagyis csak a listát, a szögletes zárójelek nélkül. Ha a szövegben szögletes zárójelre is szükségünk van, akkor ezt be kell venni a listába. Egyes szavakat, mint pl. A kasszakészlet, a kiírási parancsban az eléjük írt idézőjellel is megjelölhetjük (”KASSZAKESZLET). Ezek ismeretében már előállíthatjuk a BASIC programnál már látott „szabványos” képernyőt:

KASSZAKESZLET	BEV./KI.
(Ft)	(Ft)

Az első parancssorban a kasszakészlet **:K** változójának nulla értéket adunk. Az END után visszatérünk a főprogramhoz, és az ADATBEVITEL eljárás következik:

```
TO ADATBEVITEL
 TAB 4 :K TAB 31 "''
 MAKE "E INPUT
 MAKE "K :K + :E
 TEST :E = 0
 IFTRUE STOP
 IFFALSE TEST :K < 0
 IFFALSE ADATBEVITEL
 IFTRUE HIBAJELZES
END
```

Az első parancssor a pillanatnyi kasszakészletet írja ki — amelynek az értéke pillanatnyilag nulla — balról a negyedik oszlopban, azután a kurzor a 31. oszlopra ugrik, és ott kihagy egy helyet (TAB 31”). Ott is marad a villogó kurzor, amíg be nem viszünk egy adatot. Ezt az értéket kapja az **:E** változó. A 2. parancssorban az eddigi **:K** kasszakészletet az imént bevitt **:E**-vel megnöveljük (”K = :K + :E). A 4. parancssorban ellenőrzi a program a leállítási feltételt. Ha

:E = 0 igaz, az IFTRUE visszatér a főprogramba, ahol az EGYENLEG eljárás következik. Ha a bevitt adat értéke nem nulla (IFFALSE), a program megvizsgálja, hogy a kasszakészlet még pozitív-e (TEST :K < 0). Amennyiben ez így van (IFFALSE az utolsó előtti parancssorban), az eljárás az ADATBEVITEL paranccsal ismét előhívja önmagát és következhet az új bevételezés. Ha azonban a kasszakészlet túl kicsi volt, a program a HIBAJELZES eljárást hívja meg:

```
TO HIBAJELZES
 MAKE "K :K - :E
 CR 2 TAB 16 "FIGYELEM! CR 1
 TAB 5 [A PENZTARBAN CSAK] S 1 TAB 27 :K PRINT1 [FT VAN] CR 1
 TAB 5 [ENNEL TOBB NEM VEHETO KI!] CR 2
 ADATBEVITEL
END
```

Ez az eljárás először visszaállítja a kasszakészlet előző értékét. Azután a 2., a 3. és a 4. parancssor a ”VIGYÁZAT! ... felirattal tájékoztatja a felhasználót. Az utolsó parancssorral ismét az ADATBEVITEL eljárást hívjuk.

Az EGYENLEG eljárást csak akkor hívja a főprogram, ha az ADATBEVITEL során nullát írtunk be (TEST :E = 0). Azaz ha az ADATBEVITEL 5. sorában a vizsgálat pozitív eredményt ad, visszatérünk a főprogramba, ahonnan az EGYENLEG eljárást hívjuk meg:

```
TO EGYENLEG
 U 38 "= CR 1
 PRINT [VEGOSSZEG] TAB 28 :K S 1 PRINT1 [FT] CR 1
 TAB 27 "'' U 12 "= CR 1
 TOPLEVEL
END
```

Az EGYENLEG-gel először 38 karakter hosszúságú szaggatott kettősvonalat húzunk. Alatta megjelenik a VEGOSSZEG szó és a 28-as karaktertől a kasszakészlet. A végső értéket újra aláhúzzuk, majd az END-del visszatérünk a főprogramba. A gép ott is END-et talál. Ez a második END eredményezi a program befejeződését, és a gép ismét parancs üzemmódba kerül. Ezt a kérdőjelből állapíthatjuk meg, amely mögött ott villog a kurzor. A programok begépelése meglehetősen fárasztó. Ezért a gyakran előforduló LOGO szavakat rövidíteni lehet; ez megkönnyíti a programok írását, de kevésbé áttekinthetőek lesznek.

A rövidítések használatát az ADATBEVITEL eljáráson mutatjuk be:

```
TO ADATBEVITEL
 TAB 4 :K TAB 31 "´´
 MAKE "E INPUT
 MAKE "K :K + :E
 TEST :E = 0
 IFT STOP
 IFF TEST :K < 0
 IFF ADATBEVITEL
 IFT HIBAJELZES
END
```

A LOGO-ban a program hívását is rövidíthetjük.

P. 23/a.

```
TO P
 PENZTARGEP
END
```

A program hatására, a P billentyű lenyomásakor elindul a PENZTARGEP program.

Ennek megfelelően rövidíthetjük az eljárások hívását is. Rövidítésként persze, a LOGO-alapszavakat nem szabad használni. Ha mégis használnánk, akkor hibajelzést kapunk.

Új fogalmak a LOGO-ban:

TEST, IFTRUE (IFT), IFFALSE (IFF), ERASE (ER), STOP rövidítések.

3.4 PASCAL PROGRAMOZÁS

Ismét az alapprogrammal kezdjük, amelyből hiányoznak a felhasználónak szánt információk és a kommentárok:

```
program penztargep;
var k,e:real;
begin
      k:=0;
      repeat
          writeln(k:5:2);
          readln(e);
          if k+e>=0 then k:=k+e;
      until e=0;
      writeln(k:5:2)
end.
```

Mint minden Pascal program, ez az alapprogram is két részből áll:

— a deklarációs rész, amelyben a program nevét, a változókat, azok típusát és az eljárásokat (esetünkben nincs eljárás) deklaráljuk;

— az utasítás rész, amely a tulajdonképpeni programot tartalmazza.

A változók és típusuk deklarációja fölöslegesnek tűnhet, hiszen sem a BASIC, sem a LOGO nem igényli ezt. Nagyobb programok kifejlesztésénél azonban sok tartalmi hibát fedhet fel.

Először a kassza kezdeti értékét nullára állítjuk (K:=0). Az utasítások egy ciklusban (REPEAT ... UNTIL) addig ismétlődnek, amíg a bevitt adat nulla nem lesz (UNTIL e:=0).
A gép kiírja a kassza pillanatnyi tartalmát (WRITE (k);), majd egy adat bevitelét várja READLIN (e);). Ha ezzel az egyenleg nem lesz negatív (k+e> = 0),

akkor (THEN) a program kiszámítja az új egyenleget (k := k + e). Majd visszaugrik a REPEAT utáni első utasításhoz, ha a bevitt adat nem volt nulla (UNTIL e := 0).

Ha azonban az egyenleg kisebb nullánál, a THEN-t követő összegzést nem hajtja végre, hanem az UNTIL sorra ugrik, és folytatódik a ciklus.

Ha elhagyja a ciklust (e := 0), akkor kiírja a képernyőre az utolsó kiszámított egyenleget.

Ha lefuttatja a programot, most is, mint minden ilyen alapprogram esetén, kifogásolhatja a meglehetősen silány képernyőfelépítést. A 3.1 alfejezetben bemutatott képernyőterv adatait beadva, kb. ezt kaphatja:

```
0.00
8
8.00
7
15.00
-16
15.00
-12
3.00
0
3.00
>
```

vagy az OXFORD—Pascal-ban

```
ready
```

Ezért oldjuk meg úgy a feladatot a Pascalban is, hogy a programot kiegészítjük a felhasználónak szánt tájékoztatókkal.

A 2. fejezetben tárgyalt térfogatszámítási feladattal ellentétben ezúttal kihasználjuk a Pascal nyelv kínálta lehetőséget: eljárásokat definiálunk. Ha a beépített eljárások, mint pl. WRITEIN, READLN stb. nem elegendőek, akkor — hasonlóan a LOGO-hoz — saját eljárásokat írhatunk, amelyek azután a kívánt feladatot végrehajtják. A LOGO-val ellentétben, ezek a parancsbővítések csak abban a programban érvényesek, amelyben megírtuk őket. Tekinthetjük persze ezeket az eljárásokat egy-egy részfeladat megoldásának is. A megoldandó feladatot először különböző részfeladatokra bontjuk, majd ezekből építjük fel a megoldást. Hosszú és bonyolult feladatok megoldásánál előbb-utóbb rá leszünk kényszerítve erre a módszerre. Egy-egy ilyen eljárást tehát részprogramként is értelmezhetünk. Ugyanúgy, mint a program, ez is deklarációs és utasításrészből épül fel. Az egyetlen különbség, hogy az eljárás a PROCEDURE szóval kezdődik a PROGRAM szó helyett.

Mint azt már említettük, az eljárásokat a Pascal programok deklarációs részében kell felsorolni, a változók és azok típusai deklarálása után.

A KEZDES, a HIBAJELZES és az EGYENLEG eljárásokkal kibővített felhasználói program a következő:

```
program penztargep;
{ A program megengedi,hogy a kasszabol tetszoleges}
{ osszegeket vegyunk ki illetve tegyunk be.Ugyan- }
{akkor biztositja,hogy hiany soha ne legyen.     }
var kasszakeszlet,bevitel:real;
  procedure kezdes;
     begin
        clrscr;
        writeln('    P e n z t a r g e p   ');
        writeln('    ===================   ');
        writeln;writeln;
        writeln(' Nulla bevitele leallitja a programot ');
        writeln;writeln;
        writeln('kasszakeszlet                bev./ki. ');
        writeln(' ( Ft )                       ( Ft )  ');
     end;
  procedure  hibajelzes;
     begin
        writeln;writeln('      Figyelem!         ');
        writeln('A penztarban csak',kasszakeszlet:10:2,'Ft van.');
        writeln(' Ennel tobb nem veheto ki.');
        writeln;
     end;
  procedure egyenleg;
     begin
        writeln('======================================');
        writeln(' v e g o s s z e g',kasszakeszlet:10:2,'Ft');
        writeln(' =================':39)
     end;
  begin { foprogram }
        kezdes;
        kasszakeszlet:=0 ;
        repeat { bevitel a kasszaba }
          write(kasszakeszlet:10:2,' ' :20);readln(bevitel);
          if kasszakeszlet+bevitel<0
                then hibajelzes
                else kasszakeszlet:=kasszakeszlet+bevitel;
        until bevitel=0;
        egyenleg
  end.
```

A program neve: PENZTARGEP. A 4. sorig lévő kommentár a programot nem befolyásolja.

A változókat **egyenleg**nek, illetve **bevitel**nek nevezzük. Bár az ilyen hosszú változónevek a program begépelésénél több munkát igényelnek, mint az egybetűs jelölések, de megkönnyítik a program elemzését, ha változtatásokra van szükség. (Ha a programot megfelelő editorral állítjuk össze, akkor a megírás közben használhatunk rövidítéseket is a hosszú változónevek helyett és csak a végén írjuk át ezeket egy helyettesítő paranccsal a tervezett megjelölésekre.)

A következő sorokban a KEPERNYO, a HIBAJELZES és a VEGALLAPOT nevű eljárásokat definiáljuk. Ezeket az eljárásokat a főprogramban hívjuk majd elő. Ekkor csak az eljárások neveit kell megadni.

A KEPERNYO nevű eljárás a program indításakor létrehozza a kívánt képernyőfelépítést. Az eljárás parancsait a 2. fejezetből már ismerjük.

A HIBAJELZES nevű eljárás is csak tájékoztatja a felhasználót. Logikai művelet ebben az eljárásban sincs. A VEGALLAPOT nevű eljárás végrehajtása során a felhasználó tájékoztatása mellett megkapjuk az eredményt is, vagyis a végső egyenleget. Az eredmény formáját kialakító utasítást a 2.4 alfejezetben már elmagyaráztuk.

A BEGIN {főprogram} sorokkal kezdődik az utasításrész. Először a KEZDES nevű eljárást hívjuk, így a felhasználó megismerheti a program használatához szükséges információkat.

Ezután az **kasszakeszlet** értéket nullázzuk.

Most következik az alapprogramból már ismert ciklus (REPEAT ... UNTIL): a program addig működik (vagyis kiírja az egyenleget és újabb adatokat kér), amíg a **bevitel** nem nulla (ez a leállítás feltétele). Minden adatbevitel után megvizsgálja, hogy az egyenleg elegendő-e a kifizetéshez. Ha ez nincs így (kasszakeszlet + bevitel <0), akkor (THEN) a HIBAJELZES eljárást hívja meg. Ellenkező esetben (egyenleg + bevitel = 0) — ez az IF utasítás ELSE-ága — a program elfogadja a **bevitel**t, és kiírja az új **egyenleg**et, amely a régi **kasszakeszlet** és a **bevitel** összegéből adódik (kasszakeszlet: = kasszakeszlet + bevitel).

Ha a bevitel nulla, akkor kilépünk a ciklusból (UNTIL bevitel = 0). Ezt követi a program utolsó utasítása, a végállapot nevű eljárás, és ezzel a program befejeződik.

Az alapprogramban az IF *feltétel* THEN *utasítás* alakú feltételes elágazást használtuk, amely akkor hajtja végre a THEN utáni utasítást, ha a feltétel teljesül, egyébként folytatja a programot. A részletes programban az IF *feltétel* THEN *utasítás* ELSE *utasítás* formájú feltételes elágazást alkalmaztuk. Ha a feltétel teljesül, akkor itt is a THEN-t követő utasítást hajtja végre a program, majd folyatódik. De ha a feltétel nem teljesül, akkor az ELSE-t követő utasítás végrehajtása után folytatódik a program.

Ha összehasonlítjuk a feladat megoldását LOGO-ban és Pascalban, párhuzamot vonhatunk:

LOGO	**Pascal**
KEZDES	KEZDES
ADATBEVITEL	FOPROGRAM
HIBAJELZES	HIBAJELZES
EGYENLEG	EGYENLEG

Az eljárások szerepe a két programban gyakorlatilag azonos. A két nyelv lehetőségeit figyelembe véve még több szempontból is összehasonlíthattuk volna a megoldásokat. Mellőzhettük volna pl. a LOGO-ban a segédeljárások definiálását, illetve bevezethettük volna ugyanezeket a Pascalban is. Pascalban az eljárásokat a változók deklarálása után kell definiálnunk.

Új fogalmak a Pascalban:

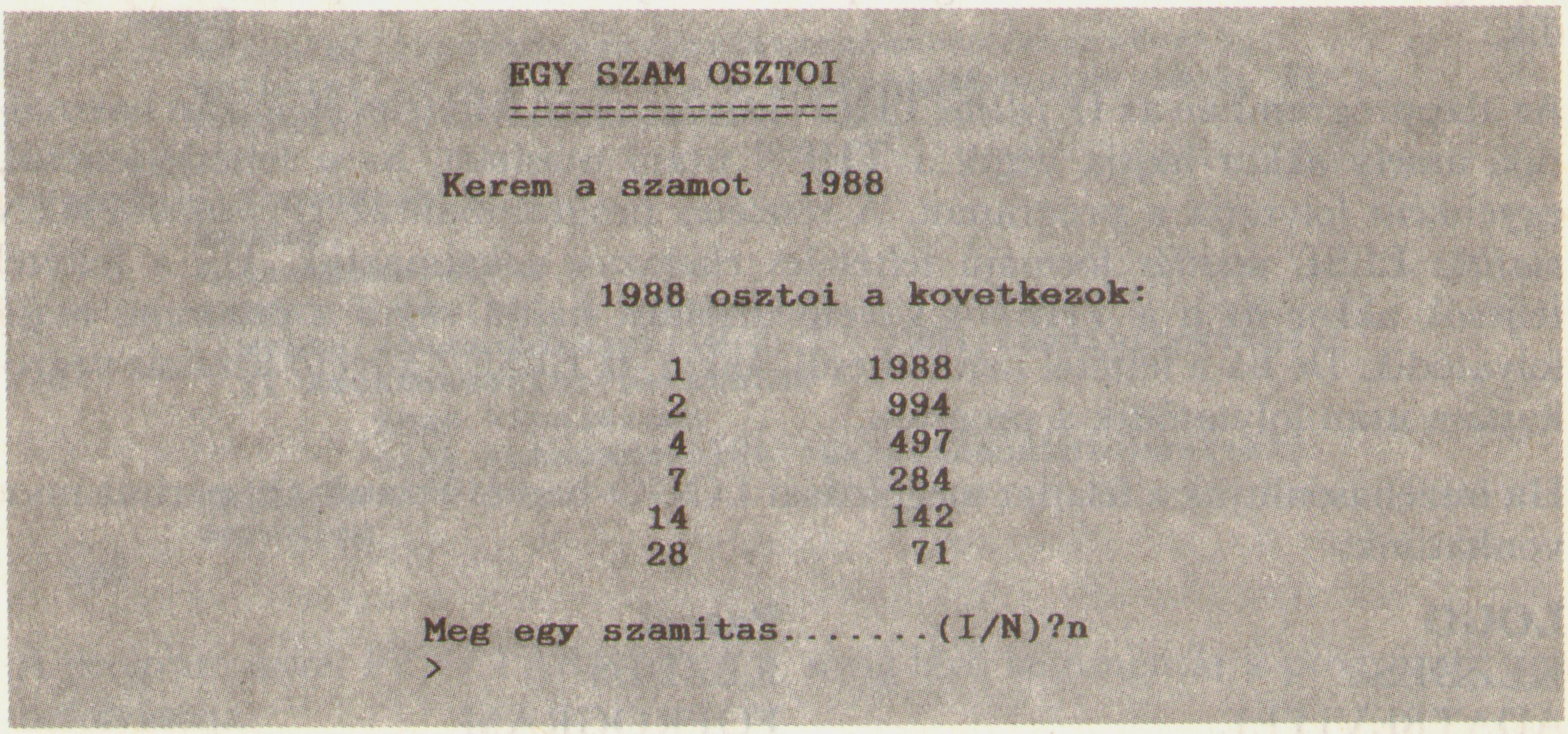
PROCEDURE, REPEAT ... UNTIL, IF ... THEN ... ELSE.

4. KERESD AZ ÖSSZES OSZTÓT!

4.1 A FELADAT MEGHATÁROZÁSA

Egy adott szám osztóit kell meghatározni és kiíratni. Kérdezze meg a program, hogy melyik számról van szó. Ha az osztók kinyomtatása befejeződik, kérdezze meg, hogy szükséges-e további számítás.

A képernyő felépítését, amelyet mind a három programnyelven elő akarunk állítani, az 576-os szám példáján mutatjuk be:



```
                  EGY SZAM OSZTOI
                  ===============

              Kerem a szamot  1988


                        1988 osztoi a kovetkezok:

                           1         1988
                           2          994
                           4          497
                           7          284
                          14          142
                          28           71

             Meg egy szamitas.......(I/N)?n
             >
```

Ahogy ez a képernyőn látszik, az osztókat nem nagyság szerint íratjuk ki, hanem minden egyes osztóhoz, vele egy sorban, megadjuk a kiegészítő osztóját. Ez lerövidíti a futási időt, és egyszerűbbé teszi a programot.

A három nyelv egyikében sincs olyan parancs, amely közvetlenül megadná, hogy egy szám osztható-e egy másik számmal vagy nem. Ha meg akarjuk tudni, hogy a 17 osztója-e a 223-nak, ezt csak kerülő úton tudjuk megállapitani. Kiszámítjuk a 223:17-et, ha az eredmény egész szám — vagyis nem tizedestört — akkor a 17 osztója a 223-nak.

Hogyan tudhatjuk meg, hogy az eredmény tizedestört-e? E kérdés megválaszolására a három nyelv különféle parancsokat kínál. Ezekkel dönthetjük el, hogy az osztás maradék nélküli, avagy sem. A LOGO-ban közvetlenül a maradékot kérdezhetjük meg. A BASIC-ben és a Pascalban azonban a számot egészrészre és tizedestörtrészre kell bontanunk:

223:17 → eredmény = 13,1176...
egészrész (223:17) = 13
eredmény (223:17) ≠ egészrész (223:17)
tehát a 17 nem osztója a 223-nak
357:17 → eredmény = 21
egészrész (357:17) = 21
eredmény (357:17) = egészrész (357:17)
tehát a 17 a 357-nek osztója, a 21 pedig a 17 kiegészítő osztója.

4.2 KIDOLGOZÁS BASIC-BEN

Az alapprogramban el kell végeznünk:
- a vizsgálandó szám megkérdezését;
- egy ciklus elkészítését, amelyben a program a számot addig vizsgálja, amíg az összes osztót meg nem találja;
- a megtalált osztók kiíratását;
- megkérdezni, hogy újra lefusson-e a program más bemenő adattal.

Ez alapprogram teljesíti a kívánságunkat:

```
150 INPUT AZ
320 FOR I=1 TO SQR(AZ)
340 IF AZ/I<>INT(AZ/I) THEN 370
360 PRINT I,AZ/I
370 NEXT I
620 INPUT E$
630 IF E$="I" THEN 150
640 END

READY.
```

Látható, hogy egy FOR—NEXT ciklussal vizsgáljuk az oszthatóságot. Ebben a ciklusban az **I** ciklusváltozó rendre az 1, 2, 3, ... értékeket veszi fel egészen a bevitt **AZ** szám gyökéig. A 340-es sorban azt vizsgáljuk, hogy az **AZ** számnak osztója-e az **I.** (Az INT utasítás az AZ/I szám egészrészét veszi.) Ha **I** nem osztó, akkor a program a 370-es sorra ugrik és onnan vissza a ciklus elejére a 320-as sorra; megnövelve az **I**-t, majd a kiegészítő osztót nyomtatja ki. A program ellenőrzéséhez válasszon egy nem túl nagy számot bemenő adatként, pl. a 12-t. Így a következő képernyőábrát kell kapnia:

```
?1  12
?2  6
?3  4
?
```

Az utolsó kérdőjelre válaszként nem egy számot, hanem egy betűt vár. Esetünkben ez az igen szó első betűje, az „I". A BASIC két különböző változótípust ismer:

az egyikben számokat tárolhatunk, ezeket egyszerűen betűkkel vagy szavakkal jelöljük (pl. A, AZ stb.); a másikban pedig karaktereket vagy karakterláncokat tárolhatunk, vagyis alfanumerikus karaktereket és ezek sorozatát. Ez utóbbit úgy jelöljük, hogy a változó neve után dollárjelet ($) írunk (pl. A$). Természetesen számokat is tárolhatunk ezekben az ún. füzérváltozókban, de ekkor nem számokként, hanem jelekként kezeljük a számokat, így nem számolhatunk velük.

A 620-as sor INPUT utasítása tehát, betűk bevitelét várja. Ha még egy szám osztóira kíváncsi, akkor I-t írjon be, egyébként egy tetszőleges más jelet. Az a felhasználó azonban, aki nem ismeri a programot, mindezt nem tudhatja, hiszen ez az INPUT parancs is csak kérdőjelet ír ki. Itt tehát különösen fontos a felhasználó tájékoztatása.

A részletes programban egyrészt a felépítést kommentáljuk, másrészt a felhasználót is megfelelően tájékoztatjuk:

```
10 REM TETSZOLEGES SZAM OSZTOINAK MEGHATAROZASA
20 REM ------------------------------------
30 REM A SZAM NAGYSAGAT CSAK A BASIC
40 REM SZAMABRAZOLASI PONTOSSAGA KORLATOZZA.
50 REM ------------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"♥":REM CLS
120 PRINT"    EGY SZAM    O S Z T O I"
130 PRINT"    ======================="
140 PRINT:PRINT
150 INPUT"ADD MEG A SZAMOT ";AZ
160 PRINT:PRINT:PRINT
170 PRINT AZ;"OSZTOI A KOVETKEZOK:":PRINT
299 :
300 REM AZ OSZTOK MEGHATAROZASA ES KIIRATASA
310 SS=INT(SQR(AZ))
320 FOR I=1 TO SS
330 TL=AZ/I
340 IF TL<>INT(TL) THEN 370
350 IF I=TL THEN PRINT TAB(4);I:GOTO 370
360 PRINT TAB(4);I;TAB(17);TL
370 NEXT I
599 :
600 REM VEGE
610 PRINT
620 INPUT"MEG EGY SZAMITAS?       (I/N)";EI$
630 IF EI$="I" THEN 100
640 END

READY.
```

Az utasítások a 110-es sorban szokás szerint a képernyő törlésével kezdődnek. A 120-as sor a címet íratja ki, amelyet a 130-as sorban aláhúz. A 140-es sorral két üres sort hoz létre a kpernyőn, majd a 150-es sorban megkérdezzük, melyik szám osztóit kell meghatározni. Ez lesz az **AZ** változó értéke. Három üres sor a képernyőn, majd a 170-es sorban kiírja, hogy a „... osztói a következők:". A számítási rész a 300-as sorban kezdődik. Mivel az osztókkal együtt a kiegészítő osztókat is kinyomtatjuk, elegendő az osztókat a szám négyzetgyökéig keresni. A szám négyzetgyökét az SQR() függvénnyel számítjuk ki (SS = INT (SQR(AZ))). Eddig vizsgáljuk majd az osztókat.

A 320-as sorban indul a FOR—NEXT ciklus, amelynek a felépítését az alapprogramban már ismertettük. Azért, hogy az **AZ/I**-t ne kelljen kétszer kiszámítani (mint az alapprogramban), az eredményt a **TL** változóba helyezzük (330-as sor). A FOR—NEXT ciklus befejezése után megkérdezzük, hogy szükséges-e még egy számítás. Válaszként az I és az N betűt adjuk meg. Ha az I-t gépeljük be, a 630-as sor visszaugratja a programot a kezdéshez, bármely más karakterre véget ér a program.

Új fogalmak a BASIC-ben:

FOR—NEXT, INT, SQR.
Ezenkívül először használunk a programban numerikus változók mellett füzérváltozókat, amelyeket a BASIC-ben a változó neve mögé írt dollárjellel ($) jelölünk.

4.3 MEGOLDÁS LOGO-BAN

Az alapprogramhoz tudnunk kell, hogyan lehet a LOGO-ban az oszthatóságot vizsgálni. Ezt a REMAINDER függvénnyel tehetjük meg: REMAINDER :A :B értéke csak akkor nulla, ha B osztója A-nak, A program végén megkérdezzük, hogy kívánunk-e új számítást. Ehhez olyan változót használunk, amelyet jellel látunk el. A LOGO ugyanis nem tesz különbséget a karakteres és numerikus változók jelölése között. A változó típusa mindig a behelyettesítésre kerülő konstanstól függ. Azt, hogy egy változó milyen típusú volt, a program futása során a .TYPE változónév paranccsal kérdezhetjük le. Egész típusú változó esetén 3-as értéket ad, lebegőpontosra pedig 4-est.
A LOGO-ban a legnagyobb egész érték a 2 130 000 000 (C 64-esre). Sajnos ilyen nagy számokra nem tudjuk egy lépésben lefuttatni a programot, mert nagyon hamar betelik a tárterület. Ilyenkor a megkapott legkisebb kiegészítő osztóval újra indítható a program, de ne felejtsük el előtte kiadni az ERNAME parancsot! (Megjegyzés: ennél nagyobb lebegőpontos számokra „bolondul" meg a BASIC nyelven írt program.)
A ciklus leállítási feltételeként használhatnánk a BASIC-hez hasonlóan az SQRT függvényt. Mivel az osztó nem lehet nagyobb, mint a kiegészítő osztója, ez az összefüggés itt felhasználható a számítás befejezésére is. Így kicsit másképp járunk el, mint a BASIC programban. Ahhoz, hogy a számítási eljárás többször előhívható legyen, az alapprogramot két eljárásból állítjuk össze:

```
TO KEZD
 MAKE "N FIRST RQ
 MAKE "T 0
 SZAMOLJ
 MAKE "E FIRST RQ
 IF :E = "I THEN KEZD
END
```

```
TO SZAMOLJ
 MAKE "T :T + 1
 IF REMAINDER :N :T = 0 THEN PRINT1 :T PRINT1 [.....] PRINT1 :N / :T CR 1
 TEST :T > :N / :T
 IFF SZAMOLJ
 IFT STOP
END
```

A KEZD eljárásban megadjuk azt a számot, amelynek az osztóit keressük. Ezenkívül szükségünk van egy számlálóra is, ennek kezdeti értéke nulla. Ezután előhívjuk a SZAMOLJ eljárást, amely tulajdonképpen a számítást végzi. Ha a számítás befejeződik, akkor visszatér a KEZD programba és egy újabb adat bevitelére vár (MAKE "E FIRST RQ). A bevitt adattól függően vagy újraindul a KEZD program, vagy a program futása befejeződik.

Most nézzük meg a SZAMOLJ eljárást.
Először a számláló (:T) értékét eggyel növeljük. Ezután a REMAINDER függvénnyel vizsgáljuk meg az oszthatóságot. Önöknek nyilván meglehetősen szokatlan a **REMAINDER :N :T** írásmód. A BASIC-ben és a Pascalban az operátorok általában az operandusok között találhatók. A Pascalban például egy **N** számnak egy **T** számra vonatkozó maradékát, vagyis, hogy az **N** szám **T**-vel osztva mennyit ad maradékul, a MOD függvénnyel számítjuk ki (N MOD T). A LOGO az ún. *lengyel formát* használja. Itt először a műveletet adjuk meg, és csak azután a számokat, amelyekre művelet vonatkozik. A művelet meghatározott számú változóra alkalmazható. Ha ennél több vagy kevesebb változónk van, akkor a műveletet és a hozzá tartozó változókat kerek zárójelbe tesszük. Ha a maradék zérus, akkor a program az osztót (:T) és a kiegészítő osztót (:N/:T) kiírja a képernyőre. Ezután az ellenőrzés következik, hogy véget ért-e az osztók kiszámítása. Ehhez azt kell megvizsgálni, hogy az osztó (esetünkben a :T számláló) nagyobb-e már a kiegészítő osztónál (:N / :T). Ha ez a feltétel nem teljesül (IFF), újra a SZAMOLJ eljárást hívjuk, amit a gép **(:T + 1)**-gyel fog megismételni. Ha az osztó már nagyobb a kiegészítő osztónál (IFT), akkor a STOP paranccsal térünk vissza a KEZD eljáráshoz, ahol az I billentyű lenyomásával megismételtethetjük a programot (IF :E = "I THEN KEZD), különben a program végére érünk.

Kibővítjük a programot a felhasználónak szóló utasításokkal. Ehhez a SZAMOLJ számítási eljárást gyakorlatilag változatlanul hagyjuk, írunk azonban egy új indító eljárást OSZTO néven, és három eljárást a felhasználói információknak.

```
TO OSZTO
 OSZTOKEZD
 OSZTOK
 KERDES
END
```

```
TO OSZTOKEZD
 ND CR 1
 TAB 5 [EGY SZAM OSZTOI] CR 1
 TAB 5 "'' U 23 "= CR 3
 PRINT [KEREM A SZAMOT !] CR 2
 MAKE "N FIRST RQ
 CR 3 ( PRINT :N [SZAM OSZTOI A KOVETKEZOK:] )
 CR 1 MAKE "T 0
END

TO OSZTOK
 MAKE "T :T + 1
 IF REMAINDER :N :T = 0 THEN KIIRAS
 TEST :T > :N / :T
 IFFALSE OSZTOK
 IFTRUE STOP
END

TO KIIRAS
 ; ELKERULJUK,HOGY NEGYZETSZAM ESETEN A NEGYZETGYOKOT KETSZER KINYOMTASSUK
 TEST :T = :N / :T
 IFTRUE TAB 4 :T CR 1
 IFFALSE TAB 4 :T TAB 17 :N / :T CR 1
END

TO KERDES
 CR 2 PRINT [MEG EGY SZAMITAS(I/N)] CR 1
 MAKE "E FIRST RQ
 IF :E = "I THEN OSZTO
END
```

Az OSZTOKEZD programmal a felhasználót tájékoztató képernyőfeliratokat készítjük el: kiírjuk, hogy egy szám osztóit keressük, és kérjük, hogy adjon meg egy számot. A beírt szám — ugyanúgy mint az alapprogramban — lesz az **:N** változó értéke. Három sort kihagyva, újabb felirat jelenik meg a képernyőn:

A ... SZAM OSZTOI A KOVETKEZOK:

Ezután a ciklusváltozó (:T) értékét nullára állítjuk. Ezzel visszatérünk a főprogramba és előhívjuk az OSZTOK számítási eljárást. Ez lényegében azonos az alapprogrambelivel. Különbség csak annyi, hogy az osztókat a KIIRAS eljárással íratjuk a képernyőre. Ez utóbbi eljárásban minden egyes osztót összehasonlítunk a kiegészítő osztójával (TEST :T = :N / :T), és azonosak, akkor csak az egyiket íratjuk ki, ha nem, akkor mind a kettőt. Ha megvan az összes osztó (:T > :N / :T), a program visszatér a főprogramhoz és a KERDES eljárással folytatódik. Igenlő válasz esetén a program ismét elölről indul, egyébként befejeződik.

Új fogalmak a LOGO-ban:

REMAINDER :A :B, IF—THEN, lengyel forma.

4.4 PASCAL PROGRAMOZÁS

Az alapprogramban megválaszolásra váró kérdések:

- hogyan állapíthatjuk meg az osztókeresés folyamatának határát;
- mi az oszthatóság feltétele, és
- hogyan tehetjük fel a számítás ismétlésére vonatkozó kérdést.

A számításhoz három változó kell:

- a szám, amelynek az osztóit keressük;
- az osztó és a
- kiegészítő osztó.

A szám, az osztója és a kiegészítő osztó mindegyike egész szám. Ennek ellenére valós típusúnak deklaráljuk, mert a Pascalban az egész típusú (INTEGER) számok nem lehetnek nagyobbak mint 2^{15}; így csak 32 768-nál kisebb számot vizsgálhatnánk. A számítás megismétlésére vonatkozó kérdéshez karakter típusú változó kell. A BASIC-hez hasonlóan az oszthatóság vizsgálatára az egészrész függvényt (INT) használjuk. A Pascalban is van maradék függvény (A MOD B), de ez csak az egész típusú számokra alkalmazható, s ezért — az előzőekben említett ok miatt — céljainknak nem felel meg. A program megszakításának feltétele, csakúgy mint a LOGO-ban, az, hogy az osztó sosem lehet nagyobb a kiegészítő osztójánál. Így a következő alapprogramot kapjuk:

```
program oszto;
    var z,t,k:real ;
    jel       :char;
 begin
    repeat
       readln(z);
       k:=z;
       t:=1;
       while t<k do
          begin
             k:=z/t;
             if k=int(k) then writeln(t,' ',k);
             t:=t+1
          end;
          readln(jel);
    until  jel<>'i'
 end.
```

Figyelje meg a 3.4 alfejezetben már alkalmazott REPEAT ... UNTIL ciklust! A REPEAT és UNTIL közötti utasításokat a program addig hajtja végre, amíg I-t gépelünk be. Mivel a kérdés (UNTIL JEL < >'I') a végén szerepel, ezeket az usatísásokat legalább egyszer végrehajtja. Ezzel ellentétes a WHILE—DO ismétlő ciklus, amellyel az osztókat számítjuk ki. Itt az elején vizsgáljuk meg a feltételt. Ha a feltétel már elsőre sem teljesül (pl. 0 osztóinak kiszámítása), a program a ciklus egyetlen utasítását sem hajtja végre. Ekkor azonnal a belső ciklus vége után folytatódik a program. (A WHILE—DO ciklushoz tartozó utasításokat BEGIN ... END közé kell írni, ha egynél több van belőlük.)

Egyébként meghatározhatnánk az osztókat a BASIC programnak megfelelően is. Figyelni kell azonban arra, hogy a BASIC-ben a négyzetgyök függvényt SQR()-rel jelölik, míg a Pascalban SQRT()-vel.

Az alapprogram, mint eddig még mindig, megoldja ugyan a feladatot, de a megoldás sem a felhasználónak, sem a megfigyelőnek nem áttekinthető. A kibővített változatban mindkét szempontot figyelembe véve úgy építjük fel a képernyőt, mint ahogy ezt a feladat meghatározásánál leírtuk:

```
program Osztok;
{ Tetszoleges szam osztoinak    }
{ meghatarozasa. A szam nagysa- }
{ gat csak a Pascal rendszerben }
{ a valos szamok helyi erteke-  }
{ nek mennyisege korlatozza.    }
var szam,oszto,
    osztoparja  :real;
    jel         :char;
procedure keppernyobevitel;
   begin
       clrscr;
       writeln('    EGY SZAM OSZTOI    ');
       writeln('    ===============    ');writeln;
       write('Kerem a szamot');
       readln(szam);
       { Bevitel. }
   end;
procedure osztoszamitas;
{ Osztok keresese es kiirasa. }
   begin
       writeln;writeln;
       writeln(szam:12:0,'osztoi a kovetkezok:');writeln;
       osztoparja:=szam;
       oszto:=1;
       while oszto<osztoparja do
          begin
              osztoparja:=szam/oszto;
              if osztoparja=int(osztoparja) then
                 if osztoparja=oszto
                    then writeln(oszto:12:0)
                    else writeln(oszto:12:0,osztoparja:12:0);
              oszto:=oszto+1
          end;
   end;
begin { Foprogram. }
   repeat
       kepernyobevitel;
       osztoszamitas;
       writeln;
       write('Meg egy szamitas......(I/N)?');read(jel);
   until (jel<>'i') and (jel<>'I')
end.
```

A feladatot kisebb részfeladatokra bontjuk (ugyanúgy, mint a 3. fejezetben) és ezeket a részfeladatokat eljárások segítségével oldjuk meg. Így a főprogram rövid és áttekinthető lesz. Csak két eljárást készítünk: a képernyőfelépítésére és az osztók kiszámítására. S a főprogram végén tesszük fel a kérdést, hogy fusson-e a

program újra, avagy sem. Mivel a legtöbb számítógép ismeri a kis- és nagybetűket is, a program leállítására vonatkozó feltételt I-vel is és i-vel is megadhatjuk. Erre az AND logikai operátort használjuk. A változókat úgy jelöljük, hogy a jelölés emlékeztessen a változó programbeli szerepére, esetünkben SZAM, OSZTO, OSZTOPARJA, JEL.
A KEPERNYOBEVITEL eljárás, a képernyő törlése után (CLRSCR) kiírja a program címét és kéri a vizsgálandó szám megadását. Az OSZTOSZAMITAS eljárás oldja meg a másik részfeladatot: elvégzi a számolást és kiírja az osztókat. Első osztóként az 1-et és kiegészítő osztóként magát a számot adjuk meg. Így a WHILE—DO ciklus leállási feltételét a WHILE OSZTO < OSZTOPARJA DO alakban fogalmazhatjuk meg. A cikluson belüli utasítások az alapprogramnak megfelelően épülnek fel.
Minden ciklusban rendkívül fontos, hogy a leállás feltétele elérhető legyen. Képzelje el, mi történne ha kifelejtené az OSZTO := OSZTO + 1 utasítást; végtelen ciklus jönne létre.
A REPEAT—UNTIL ciklust a felhasználó tudja leállítani, a cikluson belül elhelyezett kérdésre válaszolva. Ameddig I-vel (vagy i-vel) válaszol a kérdésre, addig fut le újra és újra a program.

Összehasonlítva a Pascal és a LOGO programokat, megállapíthatjuk, hogy a megoldáshoz vezető lépések más és más eljárások voltak. Itt is lenne arra lehetőség, hogy mindkét megoldási módot közvetlenül átírjuk a másik nyelvre. Ez érvényes az oszthatóság megállapítására is: a LOGO REMAINDER függvényének a Pascal MOD függvénye felel meg.

Új fogalmak Pascalban:

WHILE—DO, karakteres változó (CHAR), beépített függvények (INT és SQRT), operátorok (MOD és AND).

5. KAMATOS KAMAT SZÁMÍTÁSA

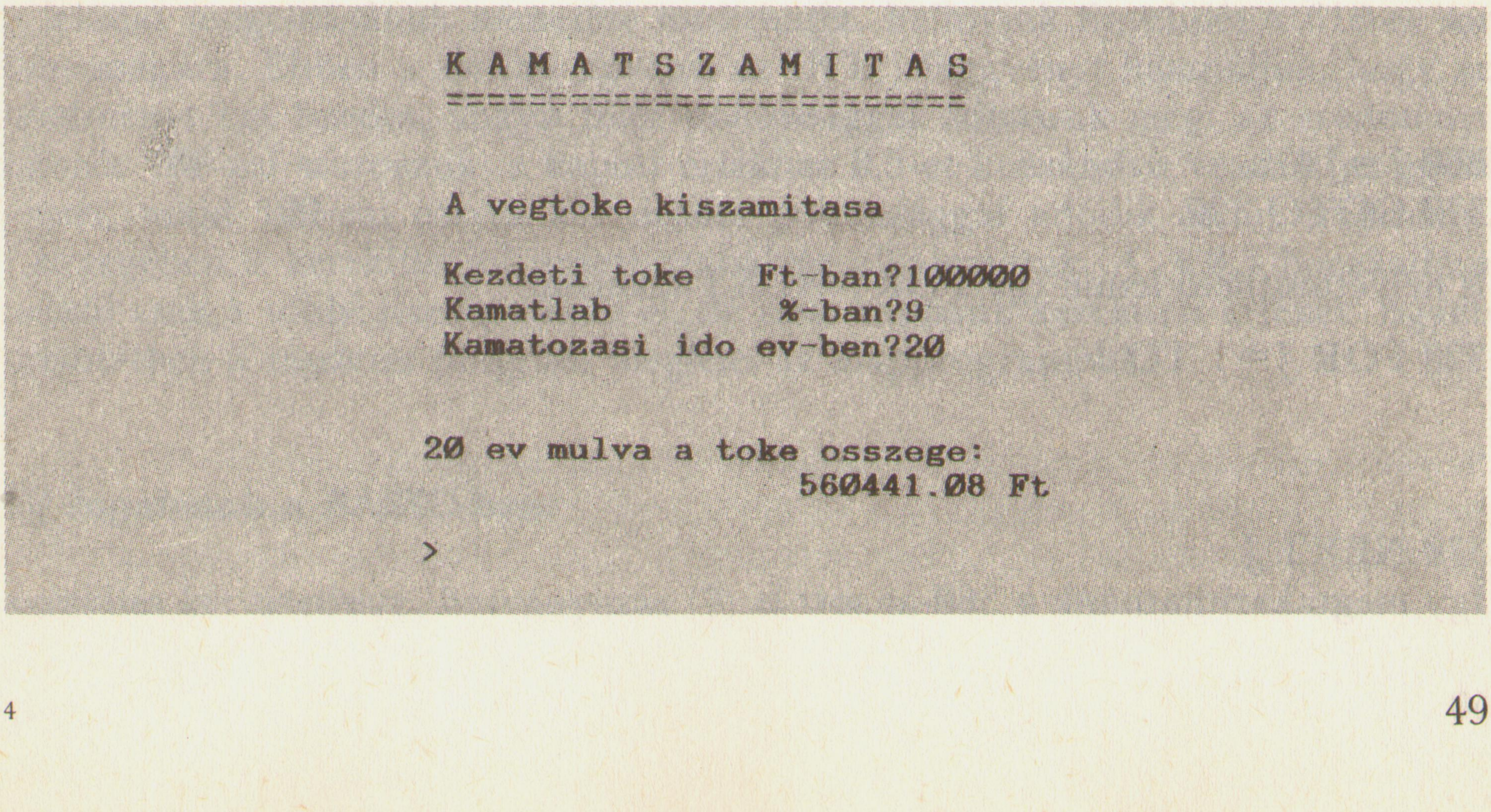
5.1 A FELADAT MEGHATÁROZÁSA

A program a kamatszámítás egyik alapfeladatát oldja meg: kiszámítja a tőkenövekedést az alaptőkéből, a kamatlábból és a kamatozás idejéből. A számításnál csak egész éveket veszünk figyelembe, hogy ne legyen túl hosszú a program. Az év végén a kamatot hozzáadjuk a tőkéhez — tőkésítjük —, azaz kamatos kamatot fogunk számolni. Ennek megfelelően az alábbi képletet alkalmazhatnánk:

$$VT=KT*(1+K/100)\uparrow I$$

(a képletben szereplő ↑ jel a hatványozás jele),
ahol:
VT a végtőke;
KT a kezdeti tőke;
K a kamatláb százalékban;
I a kamatozás ideje.

Hogy mégis más utat választunk, annak két oka van: egyrészt a következő fejezetben szükségünk lesz a közbülső számítási eredményekre, másrészt programtechnikailag sokkal világosabb, ha bemutatjuk a kamatos kamat kiszámításához szükséges algoritmust. Ezért kiszámítjuk az évenkénti kamatot, amit év végén tőkésítünk, majd az új tőkével hajtjuk végre a számításokat.
A következő ábrán a tervezett képernyőfelépítést mutatjuk be:



```
K A M A T S Z A M I T A S
=========================


A vegtoke kiszamitasa

Kezdeti toke     Ft-ban?100000
Kamatlab           %-ban?9
Kamatozasi ido ev-ben?20


20 ev mulva a toke osszege:
                  560441.08 Ft

>
```

5.2 KIDOLGOZÁS BASIC-BEN

Nagyon egyszerű alapprogram írható a kamatos kamatra vonatkozó képlet felhasználásával:

```
160 INPUT KT,K,I
640 PRINT KT*(1+K/100)^I

READY.
```

De amit azt már említettük, más megoldást választunk. Így az alapprogram valamivel terjedelmesebb lesz:

```
160 INPUT KT,K,I
310 J=0
320 IF J>=I THEN 640
330 PW=KT*K/100
340 KT=KT+PW
360 J=J+1
370 GOTO 320
640 PRINT KT

READY.
```

A 330-as sorban számítjuk ki az éves kamatot, amelyet a 340-es sorban adunk hozzá a kezdeti tőkéhez. A következő menetben (GOTO 320) már a megnövelt tőkével számoljuk ki a kamatot.
Minden menetben a 320-as sor vizsgálja meg, hogy letelt-e már a kamatozás ideje. A vizsgálathoz a **J** évszámlálót használjuk, amely értéke minden ciklusban 1-gyel nő (360-as sor). A kamat kiszámítása és hozzáadása a tőkéhez (330-as és 340-es sor) összevonható egy utasítássá:

330 KT=KT*(1+K/100)

Így kiesne a 340-es sor, de ugyanakkor az évi kamat közbülső értékei sem állnának rendelkezésre.
Ugyanígy a ciklus
320 IF J=I THEN 640
.
.
.
360 J=J+1
370 GOTO 320

is helyettesíthető lenne egy FOR—NEXT ciklussal

320 FOR J=1 TO I
.
.
.
370 NEXT J
Így megtakaríthatnánk a 360-as sort is.

Ha ehhez az alapprogramhoz még alkalmazói tájékoztatásokat és kommentárokat is fűzünk, megkapjuk a feladat teljes megoldását:

```
10 REM KAMATSZAMITAS
20 REM -----------------------------------------------
30 REM A VEGTOKE KISZAMITASA ADOTT KEZDETI TOKEBOL,
40 REM KAMATLABBOL ES KAMATOZASI IDOBOL.A KAMATOZASI IDO
50 REM CSAK TELJES EVEKBEN ADHATO MEG!
60 REM -----------------------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"▯":REM CLS
120 PRINT"     K A M A T S Z A M I T A S"
130 PRINT"     ========================="
140 PRINT:PRINT
150 PRINT"VEGTOKE KISZAMITASA":PRINT
160 INPUT"KEZDETI TOKE FT-BAN  ";KT
170 INPUT"   KAMATLAB  %-BAN  ";K
180 INPUT"KAMATOZASI IDO (EV)  ";I
190 K=K/100
200 GW=KT
299 :
300 REM AZ EVI KAMATOK KISZAMITASA
310 J=0 :REM EVSZAMLALO
320 IF J>=I THEN 600
330 PW=GW*K
340 GW=GW+PW
360 J=J+1
370 GOTO 320 :REM KOVETKEZO EVRE A SZAMOLAS
599 :
600 REM KIIRATAS ES VEGE A PROGRAMNAK
610 VT=INT(GW*100+.5)/100 :REM KEREKITES 2 TIZEDESJEGYRE
620 PRINT:PRINT
630 PRINT J;"EV MULVA A TOKE OSSZEGE:"
640 PRINT TAB(20) VT;"FT"
650 END

READY.
```

Az alapprogramban a kamat értékét (K) ciklusonként újra és újra átszámoltuk tizedestörtre (K/100). Időt takarítunk meg, ha ezt a számítást már a ciklusba érkezés előtt és csak egyszer végezzük el (190-es sor).

A program utasításait Ön már jól ismeri az eddigi fejezetekből, így teljes figyelmét a feladat megoldásának menetére fordíthatja. Felhívjuk figyelmét a 610-es sor fontosságára. Ebben a sorban ugyanis az INT() függvényt használjuk, hogy az eredményt két tizedes, vagyis fillér pontossággal adhassuk meg: az eredményt először 100-zal szorozzuk, a kapott értékhez 0,5-et hozzáadva vesszük az egész részét, majd 100-zal osztva két tizedes pontossággal adódik az új eredmény. Lényegében a kerekítés jól ismert szabályát alkalmaztuk. A +0,5 hozzáadása nélkül csak a többi tizedes tűnne el, de nem kapnánk kerekített értéket. Egyéb tekintetben a részletes program megfelel az alapprogramnak.

Új fogalmak a BASIC-ben:

kerekítés két tizedesre, hatványozás (↑).

5.3 MEGOLDÁS LOGO-BAN

Az alapprogramot két eljárásból építjük fel: az adatbeviteli és a számítási, kiírási eljárásból. A B. függelék segédparancsait is felhasználjuk az alapprogramban. Kérjük ezeket is töltse be a tárba.
Kezdjük az adatbeviteli eljárással:

```
TO A
 MAKE "T FIRST RQ
 MAKE "K FIRST RQ
 MAKE "I FIRST RQ
 R 1
END
```

Az adatbeviteli eljárás (A) során a három változónak (T — kezdeti tőke, K — kamatláb, I — kamatozás ideje) a billentyűzeten keresztül adunk értéket. Ezután hívjuk az R számítási eljárást és a számítási eljárás **:X** változója **:I** értékét kapja. (A 6.3 alfejezetben részletesebben foglalkozunk majd azzal, hogyan kell az eljárásokban szereplő változóknak értéket adni.)

Nézzük most az R számítási eljárást:

```
TO R :X
 IF :X = 0 THEN PRINT :T TOPLEVEL
 MAKE "T :T + :T * :K / 100
 R :X - 1
END
```

Az eljárásban **:X** kezdeti értéke **:I**, hiszen az előző eljárásból ezt az értéket kapta. Az eljárás mindig önmagát hívja az utolsó utsítássor miatt, de az **:X** értéke minden híváskor 1-gyel csökken (R :X − 1), egészen addig, amíg a nullát el nem éri. Ha elérte (IF :X = 0 ...), akkor kiírja **:T** (a tőke) értéket, és befejeződik a program futása (TOPLEVEL).
Választhattunk volna más számítási módot is, olyat ahol **:X** értékét növelve jutunk el a végeredményhez:

```
TO A
 MAKE "T FIRST RQ
 MAKE "K FIRST RQ
 MAKE "I FIRST RQ
 R :I
END

TO R :X
 IF :X > :L THEN PRINT :T TOPLEVEL
 MAKE "T :T + :T * :K / 100
 R :X + 1
END
```

Az első megoldás során az R eljárás változója (:X) az adatbevitel (A) eljárásban kapja meg az értékét, mégpedig a kamatozási idő (:I) értékét, a második megoldás során pedig a változó (:X) kezdeti értéke 1 lesz. Mivel **:I** értékét egy MAKE utasítással adtuk meg, így ez általános érvényű (globális) lesz, vagyis minden

további eljárásban felhasználható. A részletes megoldásban a második változatnak megfelelő számítási módot alkalmazzuk.
A programot hat részre, eljárásra bontjuk:

```
TO KAMATSZAMITAS
 KEPERNYO
 TOKE
 KAMATLAB
 IDOTARTAM
 SZAMITAS 1
 EREDMENY
END
```

A KEPERNYO eljárással feliratot helyezünk el a képernyőn, kiírjuk a program címét. (Ne felejtse el a B. függelék szerinti segédeljárásokat a gépbe tölteni!) Programtechnikailag ebben az eljárásban semmi új sincs.

```
TO KEPERNYO
 ND
 TAB 7 [KAMATSZAMITAS] CR 1
 TAB 7 "'' U 23 "= CR 3
 PRINT [A VEGTOKE KISZAMITASA]
 CR 1
END
```

A három adatbeviteli eljárás is ugyanazt a sémát követi, mint amit az utóbbi fejezetek programjainál már használtunk.

```
TO TOKE
 PRINT [KEZDETI TOKE FT-BAN?] S 1
 MAKE "TOKE FIRST RQ
END

TO KAMATLAB
 PRINT [KAMATLAB %-BAN?] S 1
 MAKE "KAMAT FIRST RQ
END

TO IDOTARTAM
 TAB 3 [IDOTARTAM EVEKBEN?] S 1
 MAKE "IDO FIRST RQ CR 2
END
```

Természetesen ezt a három eljárást egy eljárással is helyettesíthetnénk.

```
TO SZAMITAS :X
 IF :X > :IDO THEN STOP
 MAKE "TOKE :TOKE + :TOKE * :KAMAT / 100
 MAKE "TOKE ( ROUND :TOKE * 100 ) / 100
 SZAMITAS :X + 1
END
```

A SZAMITAS eljárás ugyanolyan felépítésű, mint az alapprogramban. A kiszámított értéket viszont kerekítenünk kell. Ehhez egy új LOGO parancsot használunk: a ROUND-ot. A tőkét megszorozzuk 100-zal, egész számra kerekítjük, majd az eredményt elosztjuk 100-zal. A ROUND megfelel a matematikai kerekitési szabályoknak, vagyis 5-től felfelé kerekít. (A bankokban rendszerint az ügyfél terhére kerekítenek.) A LOGO-ban a zárójelet nem úgy használjuk, mint máshol. Ennek oka a LOGO-ban használt *lengyel forma*. A zárójeleknek közre kell fogniuk a parancsot. Ez a sajátosság világosan látszik egy nyomtatási utasítás példáján.

A parancsszóval először egy szöveget, majd egy változó értéket akarunk kinyomtatni. BASIC logika szerint az alábbit írhatnánk:

PRINT ("TOKE = :TOKE)

Erre a LOGO hibaüzenettel válaszol. Írjuk most másképp:

(PRINT "TOKE = :TOKE)

Így már a gép kiírja: TOKE = 100, ha a :TOKE változó értéke 100.
Ha a gép a számítást elvégezte (IF :X> :IDŐ), visszatérünk a főprogramba. Mivel a TŐKE globális változó, minden eljárásban felhasználható, így az EREDMENY eljárásban is.

```
TO EREDMENY
 PRINT1 :IDO S 1 PRINT1 [EV MULVA A TOKE OSSZEGE] CR 2
 TAB 21 :TOKE S 1 PRINT1 [FT] CR 3
END
```

Az EREDMENY eljárás is — mint minden olyan eljárás, amellyel rendezett képernyőt kapunk — nagyrészt csak a nyomtatási formára vonatkozó utasításokból áll.

Új fogalmak a LOGO-ban:

ROUND, globális változók.

5.4 PASCAL PROGRAMOZÁS

A változók deklarálásától eltekintve a Pascal alapprogram nem sokban különbözik a BASIC-től:

```
program kamatok1;
var kt,k  :real;
    i,j   :integer;
begin
    readln(kt,k,i);
       for j:=1 to i do
       kt:=kt*(1+k/100);
    writeln(kt)
end.
```

A három értéket egymástól egy-egy üres hellyel elválasztva írjuk be, csak a harmadik szám után következik a (RETURN) billentyű lenyomása.

Itt is, mint a BASIC-ben, megírhattuk volna a programot az exponenciális függvénnyel is:

```
program kamatok2;
var kt,k :real;
    i,j  :integer;
begin
    readln(kt,k,i);
       kt:=kt*exp(i*ln(1+k/100));
       { A Pascal-ban nincs altalanos alapu exponencialis}
       { fuggveny,ezert e-alapu logaritmusra szamolunk }
       writeln(kt)
end.
```

(Megjegyzés: a Pascal nem ismeri a tetszőleges alapú exponenciális függvényt, ezért át kell írni *e* alapú exponenciális függvénnyé az ismert azonosságok szerint.)

A részletes programban nincs alapvető újdonság az eddigiekhez képest. A főprogramot a következő három eljárás alkotja: KEPERNYOBEVITEL, SZAMITAS, KIIRAS.

A BASIC és a LOGO megoldásokkal ellentétben a kerekítéshez nem kell külön parancs, mert a Pascal eleve olyan nyomtatási utasítással rendelkezik, amelyben meghatározható, hogy az eredményt két tizedes jegyre adjuk meg (...,toke:10:2,...).

A következő részletes programot kapjuk:

```
program kamatszamitas;
{ A program adott kezdeti tokebol, kamatlabbol }
{ es a kamatozasi idobol szamitja ki a vegto-  }
{ ket. A kamatozasi idot csak egesz evekben    }
{ adhatjuk meg.                                }
var kezdeti_toke,toke,kamatlab,kamatok:real;
    kamatozasi_ido,i                  :integer;
procedure kepernyobevitel;
    begin
       clrscr;
       writeln(' K A M A T S Z A M I T A S ');
       writeln(' ========================= ');
       writeln;writeln;
       writeln(' A vegtoke kiszamitasa');writeln;
       write(' Kezdeti toke   Ft-ban?');readln(kezdeti_toke);
       write(' Kamatlab        %-ban?');readln(kamatlab);
       write(' Kamatozasi ido ev-ben?');readln(kamatozasi_ido);
    end;
procedure szamitas;
var i:integer;
    begin
       toke:=kezdeti_toke;   { A kezdeti toket elrakjuk. }
       kamatlab:=kamatlab/100;
       for i:=1 to kamatozasi_ido do
          begin
             kamatok:=toke*kamatlab;
             toke:=toke+kamatok;
          end;
    end;
```

```
procedure kiiras;  { Eredmeny kiiratasa. }
    begin
       writeln;writeln;
       writeln(kamatozasi_ido,' ev mulva a toke osszege:');
       writeln('                 ',toke:10:2,' Ft ');
    end;
begin  {foprogram}
    kepernyobevitel;
    szamitas;
    kiiras;
end.
```

A FOR—TO—DO, a WHILE—DO és a REPEAT—UNTIL Pascal ciklusok GOTO-mentes programozást tesznek lehetővé. A BASIC-ben szinte nélkülözhetetlen ugrási utasításra a Pascalban gyakorlatilag nincs is szükség. Ezáltal a programok áttekinthetőbbé válnak és a hibalehetőség is kevesebb. A számítási eljárásban egy FOR—TO—DO ciklust alkalmaztunk (FOR i : = 1 TO kamatozási idő DO). Ennél a ciklusnál a ciklusváltozó (i) értéke minden egyes menetben eggyel nő, egészen addig, amíg a TO és a DO közé írt határt el nem éri. Ha egynél több utasítást akarunk ismételten végrehajtani, akkor az utasításokat, mint a példánkban is, BEGIN és END közé kell tenni. Figyelnünk kell továbbá arra is, hogy a FOR—TO—DO ciklusban a ciklusváltozó és a ciklus határa is csak skalár (integer) adattípusú lehet, de bármely sorszámozott típus megengedett.

Új fogalmak a Pascalban:

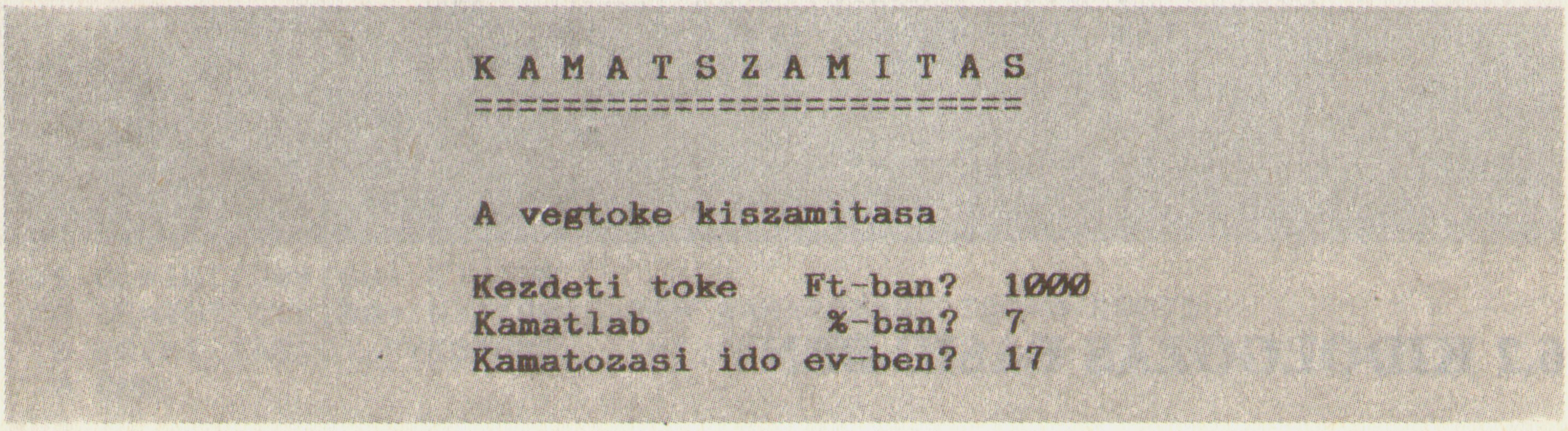
exp és ln függvények, FOR—TO—DO ciklus:
a FOR és a TO közé kell a futóindexet, a TO és a DO közé pedig a ciklus határát elhelyezni. Ezek csak skalár adattípusúak lehetnek. A futtatási utasításhoz tartozó utasítások, mint a WHILE—DO-nál, BEGIN és END közé kerülnek (ha egynél több utasításról van szó).

6. NÉHA SEGÍT EGY TÁBLÁZAT

6.1 A FELADAT MEGHATÁROZÁSA

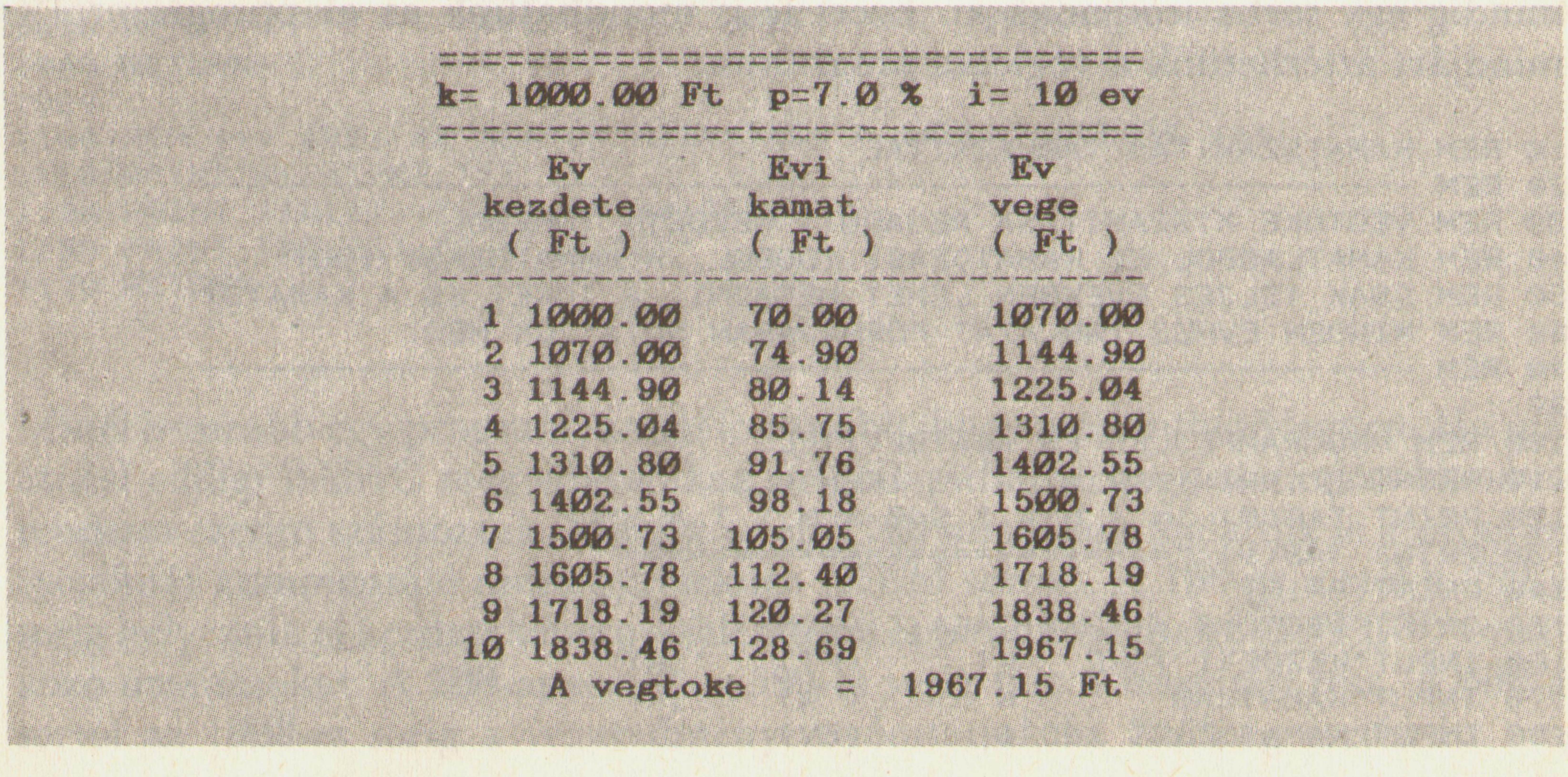
A feladat lényegében az 5. fejezet továbbfejlesztett változata; ne csak a végtőkét írja ki a program, hanem jelenjen meg a képernyőn egy négyoszlopos táblázatban az évek száma, az év eleji tőke, az éves kamat és az évvégi tőke is. A többlet kívánságok tehát a képernyőkiírásra vonatkoznak.

Mindezt úgy kell megszerkeszteni, hogy az alkalmazó könnyen olvasható táblázatot kapjon:



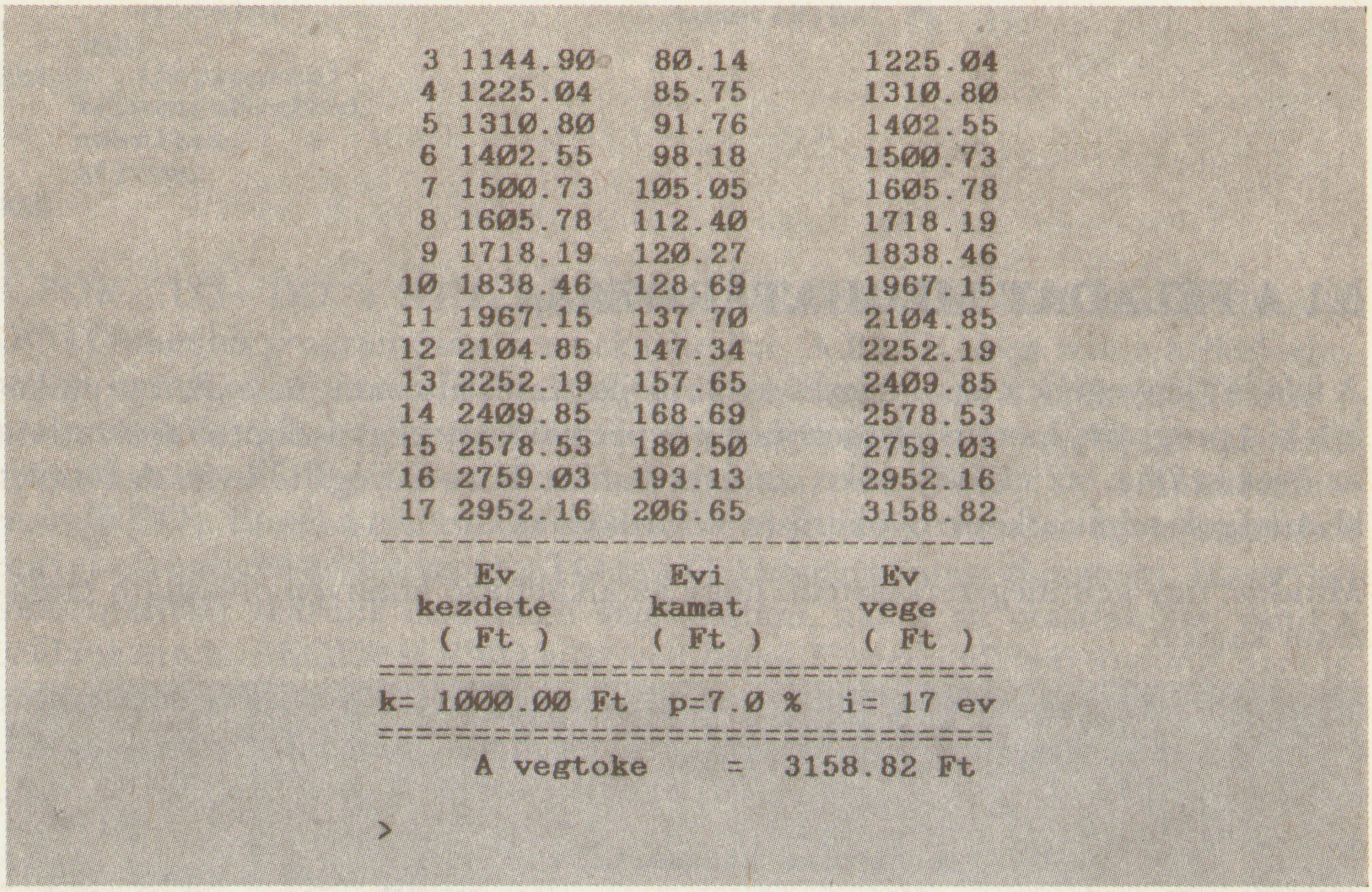
```
K A M A T S Z A M I T A S
=========================


A vegtoke kiszamitasa

Kezdeti toke    Ft-ban?  1000
Kamatlab         %-ban?  7
Kamatozasi ido ev-ben?  17
```

A képernyőábra a feladat megoldásához tervezett három képernyő közül az elsőt mutatja. Megegyezik a 5. fejezetben tárgyalt feladat képernyőfelépítésével. A k = 1000 Ft, p = 7,0 %, i = 10 év példával mutatjuk be a második képernyőábrán a táblázat kívánt megjelenési formáját:



```
================================
k= 1000.00 Ft   p=7.0 %   i= 10 ev
================================
     Ev          Evi          Ev
  kezdete       kamat        vege
   ( Ft )       ( Ft )       ( Ft )
--------------------------------
  1 1000.00     70.00       1070.00
  2 1070.00     74.90       1144.90
  3 1144.90     80.14       1225.04
  4 1225.04     85.75       1310.80
  5 1310.80     91.76       1402.55
  6 1402.55     98.18       1500.73
  7 1500.73    105.05       1605.78
  8 1605.78    112.40       1718.19
  9 1718.19    120.27       1838.46
 10 1838.46    128.69       1967.15
     A vegtoke    =    1967.15 Ft
```

Ha a képernyőre nem fér ki a teljes táblázat (pl. i = 17 év esetén), a táblázat végén még egyszer kiíratjuk a programfejet.



```
  3 1144.90   80.14      1225.04
  4 1225.04   85.75      1310.80
  5 1310.80   91.76      1402.55
  6 1402.55   98.18      1500.73
  7 1500.73  105.05      1605.78
  8 1605.78  112.40      1718.19
  9 1718.19  120.27      1838.46
 10 1838.46  128.69      1967.15
 11 1967.15  137.70      2104.85
 12 2104.85  147.34      2252.19
 13 2252.19  157.65      2409.85
 14 2409.85  168.69      2578.53
 15 2578.53  180.50      2759.03
 16 2759.03  193.13      2952.16
 17 2952.16  206.65      3158.82
--------------------------------
     Ev         Evi        Ev
  kezdete     kamat      vege
   ( Ft )     ( Ft )     ( Ft )
================================
k= 1000.00 Ft  p=7.0 %  i= 17 ev
================================
     A vegtoke    =  3158.82 Ft

>
```

6.2 KIDOLGOZÁS BASIC-BEN

A feladat alapprogramja megfelel az előző fejezet alapprogramjának, ha azt a változatot választjuk, amelyben egy ciklusban végeztettük a számítást. Csak ki kell egészítenünk a ciklus utasításait olyan parancsokkal, amelyekkel megoldhatjuk, hogy a kívánt közbülső értékek (év, év eleji tőke, éves kamat, év végi tőke) mindig egy sorba íródjanak ki. Ezért le is mondhatunk az alapprogramról és mindjárt áttérhetünk a végleges programra:

```
10 REM KAMATSZAMITAS TABLAZATTAL
20 REM ---------------------------------------------------
30 REM VEGTOKE KISZAMITASA MEGADOTT KEZDETI TOKEBOL ,
40 REM KAMATLABBOL ES KAMATOZASI IDOBOL. A KAMATOZASI IDOT
50 REM CSAK TELJES EVEKBEN LEHET MEGADNI. A TOKET ES A KAMATOT
60 REM MINDEN EVHEZ TABLAZAT FORMAJABAN MUTATJUK MEG.
70 REM ---------------------------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"□":PRINT:PRINT:      REM CLS
120 PRINT TAB(9) "K A M A T S Z A M I T A S"
130 PRINT TAB(9) "========================="
140 PRINT:PRINT
150 PRINT"VEGTOKE KISZAMITASA:":PRINT
160 INPUT"KEZDETI TOKE   KT      (FT)    ";KT
170 INPUT"KAMATLAB       K       ( %)    ";K
180 INPUT"KAMATOZASI IDO I       (EV)    ";I
```

```
190 K=K/100
200 GW=KT :REM KEZDETI TOKE MEGJEGYZESE
210 REM TABLAZATFEJ ELKESZITESE
220 PRINT"□":REM (CLS) MASODIK KEPERNYO
230 PRINT"===================================="
240 GOSUB 1100  :REM TABLAZATFEJ KITOLTESE
250 PRINT"===================================="
260 GOSUB 1000  :REM TABLAZATFEJ
270 PRINT"------------------------------------"
299 :
300 REM A KOZBENSO ERTEKEK KISZAMITASA ES KIIRATASA
310 J=0 :REM EVSZAMLALO KEZDETI ALLASA
320 IF J>=I THEN 600 :REM KESZ
330 PRINT"  ";J+1;
340 GE=INT(GW*100+.5)/100:PRINT TAB(8);GE;
350 PW=GW*K
360 GE=INT(PW*100+.5)/100:PRINT TAB(21);GE;
370 GW=GW+PW
380 GE=INT(GW*100+.5)/100:PRINT TAB(32);GE
390 J=J+1
400 GOTO 320 :REM KOVETKEZO EV KISZAMITASA
599 :
600 REM VEGE
610 REM HA A TABLAZAT HOSSZABB VOLT A KEPERNYONEL,
620 REM A TABLAZATFEJET A TABLAZAT VEGERE IS
630 REM KI KELL IRATNI.
640 IF J<10 THEN 700 :REM TABLAZAT ROVIDEBB
650 PRINT"------------------------------------"
660 GOSUB 1000  :REM TABLAZATFEJ
670 PRINT"===================================="
680 GOSUB 1100  :REM TABLAZATFEJ KITOLTESE
690 PRINT"===================================="
695 GE=INT(GW*100+.5)/100
700 GE=INT(GW*100+.5)/100  :REM KAMATOZASI IDO=0 ESETE
710 PRINT TAB(9);"VEGTOKE    =";GE;"FT"
720 END
996 :
997 :
998 REM **************** ALPROGRAMOK ******************
999 :
1000 REM *** AP1: TABLAZATFEJ KIIRATAS *******************
1010 PRINT"         EV         EVES-        EV   "
1020 PRINT"EV      KEZDETE     KAMAT       VEGE"
1030 PRINT"        (FT)        (FT)        (FT)"
1040 RETURN
1099 :
1100 REM *** AP2: AZ ADATOK KIIRATASA A TABLAZATFEJBE ***
1110 PRINT"KT =";KT;"FT";
1130 PRINT TAB(17);"K =";K*100;"%";
1150 PRINT TAB(28);"I =";I;"EV"
1170 RETURN
```

Minél hosszabb egy program, annál fontosabb, hogy áttekinthető legyen. Ha újra átnézi eddigi BASIC megoldásainkat, megállapíthatja, hogy a sorokat bizonyos rendszer szerint számoztuk. Mindig az első 100 sorszámot foglaltuk le a program tartalmát kommentáló megjegyzéseknek (REM sorok). A 100-as sortól a 299-es sorig helyeztük el kezdeti képernyőfelépítésre és az adatbevitelre (INPUT) vonatkozó utasításokat. A 300-as sortól az 599-es sorig tart a számítási rész. A 600-as sortól az 1000-es sorig a végeredményt nyomtatjuk ki. Szükség esetén itt még

egyéb feladatok megoldására is marad hely, ilyen példánkban a táblázatfej ismételt kinyomtatása.
Ebben a programban használunk első ízben alprogramokat (szubrutin), ezeket az 1000-es sortól vesszük fel a programba. A BASIC szubrutinok első ránézésre hasonlítanak a LOGO vagy Pascal eljárásokhoz, de távolról sem olyan hatékonyak. A problémák elsősorban abból származhatnak, hog a BASIC-ben minden változó globális, vagyis a programban mindenütt érvényes. Hosszú programoknál tehát rendkívüli óvatosságra van szükség, ha el akarjuk kerülni a változók nem kívánt átalakulásait.

A bemutatott programot is a most leírt rendszer szerint készítettük el. Az INPUT-rész a 200-as sorig teljesen megegyezik az 5.2 alfejezet feladatmegoldásával. A 210-es sorral kezdődik az új képernyő felépítése. A 230-as sorban két vízszintes vonalat húzunk. A 240-es sorban előhívjuk a beírt értékeket még egyszer kiíró alprogramot (GOSUB 1100). Magát az alprogramot mindig RETURN utasítással kell lezárni (1040-es, illetve 1170-es sor). A 250-es sor ismét két vízszintes vonalat eredményez. Ezután a 260-as sorban azt az alprogramot hívjuk elő, amely felírja a képernyőre az első táblázat fejlécét. Ezt a 270-es sor egyszerűen aláhúzza.
A 300-as és a 400-as sor között számítja ki a program a táblázat értékeit és azonnal mindjárt ki is írja a megfelelő helyre. Az **I**-t futóindexként és ugyanakkor évszámlálóként is használjuk. A 310-es sorban az **I**-t nullázzuk. A 320-as sorban helyeztük el a leállás feltételét. Akkor érünk a program végéhez, amikor a futóindex értéke eléri az **EV** változóban meghatározott időtartamot. A 330-as sor írja ki a **I** futóindex pillanatnyi értékét; megadja az évet. A 340-es sorban az év eleji tőkét (GW) két tizedesre kerekítve (l. 5.2 alfejezet), a táblázat második oszlopába írjuk (TAB(8);GE). Miután a 350-es sorban kiszámítja a kamatot (PW), a 360-es sorban ezt is két tizedesre kerekíti és kiíratja a táblázat harmadik oszlopába (TAB(21);GE). A 370-es sorban az év végi tőkét (GW) számítja ki, amelyet a 380-as sorban szintén kerekítünk és a táblázat negyedik oszlopába írjuk (TAB(32);GE). Ezután az **I** évszámláló értéke 1-gyel nő, és ugyanez a folyamat ismétlődik.
Mivel 16 sor kiszámítása után a táblázatfej kifut a képernyőről, de a befejezéskor látszania kell, a 640-es sorban egy feltételes elágazást helyeztünk el (640 IF I< 16 THEN 700). Ha ez nem teljesül, akkor a 660-as és 680-as sor újra a megfelelő alprogramokat hívja, amelyekkel újra kiírjuk a táblázatfejet, ezúttal a képernyő aljára. Ha **I<16** teljesül, akkor a 710-es sor a tőke végső értékét írja ki.
Egy alprogram kezdete egy BASIC programlistában csak akkor ismerhető fel, ha ezt világosan megjelöljük. Ezért az 1000-es és az 1100-as sorban egyaránt utalunk arra, hogy itt egy-egy alprogram kezdődik. Ne felejtse el, hogy minden alprogramot a RETURN szó zár le. (Nem a RETURN billentyű lenyomása!)

Ha beírta a programot, futtassa le először egyszerű értékekkel. Így hibátlan beírás esetén a második képernyőábránknak megfelelő képernyő-felépítést kap. Ha extrém értékeket választ, vagy például az időtartamot nem egész számnak adja meg, észre fogja venni, hogy a program ilyen jellegű adatok ellen nincs védve.

Ez utóbbin például így segíthetünk:

```
185 IF I — INT(I)< >0 THEN 180
```

Új fogalmak a BASIC-ben:

GOSUB—RETURN (alprogramok hívása).

6.3 MEGOLDÁS LOGO-BAN

Itt is elhagyjuk az alapprogramot. Kevesebbet kell írnia, ha az 5. fejezet eljárásait újra betölti a gépbe, néhány eljárást egy kicsit megváltoztat, másokat töröl, és csak kiegészítő eljárásokkal toldja meg a programot. A 11. fejezetből tudhatja meg, hogyan kell eljárnia.

A KEPERNYO eljárást változtatás nélkül használjuk, csak átkereszteljük TKEPERNYO-re. A TOKE, a KAMATLAB és az IDOTARTAM eljárásokból egy ADATOK nevű eljárást csinálunk úgy, hogy mind a három eljárást az editorba visszük, bizonyos sorokat törlünk vagy hozzáírunk, és végül az egész eljárást ADATOK-nak nevezzük el.

```
TO ADATOK
 PRINT1 [KEZDETI TOKE K] TAB 22 [(FT)] S 1
 MAKE "KTOKE FIRST RQ
 ; ELTESSZUK A KEZDETI TOKET
 MAKE "TOKE :KTOKE
 TAB 7 [KAMATLAB P] TAB 22 [(%-BAN?)] S 1
 MAKE "KAMAT FIRST RQ
 TAB 7 [IDOTARTAM I(EVEKBEN)?] S 1
 MAKE "IDO FIRST RQ
END
```

A SZAMITAS és az EREDMENY eljárásokra nincs szükség, ezeket töröljük a tárból az ERASE paranccsal. Ezzel is takarékoskodjunk a hellyel a tárban.

Az új főeljárásunkat KAMATTABLAZAT-nak nevezzük és a következő eljárá-

```
TO KAMATTABLAZAT
 TKEPERNYO
 ADATOK
 ND ADATFELIRAS
 TABLAZATFEJ
 U 40 "-
 SZAMITAS 1
 HOSSZABBITAS?
 EREDMENY
END
```

A főeljárásban egymás után hívjuk az eljárásokat. A TKEPERNYO kiírja nekünk a címet a képernyőre és az ADATOK eljárás során az alkalmazó beírja a számításhoz szükséges adatokat.

A további eljárások újak és most egymás után ismertetjük ezeket.

```
TO ADATFELIRAS
 U 40 "=
 PRINT1 [K =] S 1 PRINT1 :KTOKE S 1 PRINT1 [FT]
 TAB 17 [P =] S 1 PRINT1 :KAMAT S 1 PRINT1 [%]
 TAB 27 [I =] S 1 PRINT1 :IDO S 1 PRINT1 [EV] CR 1
 U 40 "=
END
```

Az ADATFELIRAS eljáráshoz először az U segédeljárást alkalmazva kettősvonalat húzunk 40 karakter szélességben. A következő három parancssor kiírja a kezdeti tőke, a kamatláb és az időtartam értékét a jeleikkel együtt. A kiírás szintén kettősvonallal zárul.

```
TO TABLAZATFEJ
 TAB 9 [EV-] TAB 21 [EVI-] TAB 33 [EV-] CR 1
 PRINT1 [EV] TAB 9 [KEZDET] TAB 22 [KAMAT] TAB 34 [VEGE] CR 1
 TAB 10 [[FT]] TAB 22 [[FT]] TAB 34 [[FT]] CR 1
END
```

A TABLAZATFEJ eljárás készíti el a négy oszlop fölött a címeket, szélességüket a várható adatokhoz igazítva.

```
TO SZAMITAS :L
 IF :L > :IDO THEN STOP
 MAKE "KAM ( ROUND :TOKE * :KAMAT ) / 100
 TAB 3 :L TAB 9 :TOKE TAB 22 :KAM
 TAB 32 ( ( ROUND 100 * ( :TOKE + :KAM ) ) / 100 ) CR 1
 MAKE "TOKE ( ROUND 100 * ( :TOKE + :KAM ) ) / 100
 SZAMITAS :L + 1
END
```

Amikor a SZAMITAS eljárást először hívjuk meg a főprogramból, az **:L** (évszámoló) értéke 1. Az eljárás a végén önmagát hívja, de **:L** értéke 1-gyel növekedik, így megfelelő számú lépésben eléri az első sorban leírt megállási feltételt (:L >:IDO). Ekkor visszatérünk a főprogramba. Ha a megállási feltétel nem teljesül, a program először a kamatot számítja ki, és két tizedesjegyre kerekíti. Ekkor a kamat százszorosával, vagyis magával a kamatlábbal számolunk, s nem P/100-zal.

Aztán egész számra kerekítünk és 100-zal elosztjuk az eredményt. Az eljárás következő sora kiírja az eltelt évek számát (:L), az év eleji tőkét és a kamatot, a sor megfelelő helyére. Mint a BASIC megoldásnál, itt is balra illesztve írjuk be az oszlopokba a számokat, tehát számításba kell venni, hogy a szám végéről a nullák lemaradhatnak. (Sem a LOGO, sem a standardnak tekintett BASIC nem tartalmaz olyan formátum utasítást, amely az egyes oszlopokban mindig ugyanarra a helyre teszi ki a tizedesvesszőt vagy tizedespontot.) A következő —

negyedik — parancssor az év eleji tőkéből és a kamatból kiszámítja az új tőkét, majd a táblázat negyedik oszlopában kiírja (TAB 32 :TOKE CR 1). Végül újra előhívja a számítás eljárást úgy, hogy az évszámlálót (:L) eggyel növeli.

Azokat a változókat, amelyeket csak egy eljáráson belül használhatunk és más programrészekben általában nem hívhatók elő, *lokális* változóknak nevezzük. Amikor meghívjuk az eljárást, vele együtt jön az eljárás változója is. Ha kilépünk az eljárásból és a fölérendelt eljárásban folytatjuk a munkát, a gép törli a változó értékét. Ezért nem okozhat bajt, ha ugyanazt a változónevet a program más részén más tartalommal használjuk. *Globális* változóknak nevezzük azokat a változókat, amelyeknek — a lokális változókkal ellentétben — az értéke a program bármely helyén használható. Minden olyan változó, amelynek a MAKE ”... utasítással adunk értéket, globális változó. Egy eljárás lokális változói minden fölérendelt (hívó) eljárás számára ismeretlenek, viszont minden alárendelt eljárásban rendelkezésre állnak. Elmondhatjuk tehát, hogy a lokális változók az alárendelt eljárásokra globálisak.
A SZAMITAS eljárásban az **:L** lokális változó. Minden lokális változó nevét az eljárás neve után fel kell tüntetnünk (TO SZAMITAS :L). Az eljárás hívásakor (SZAMITAS 1) kap értéket ez a változó, így a kezdeti értéke 1 lesz. Az értéket változóval is átadhatnánk:

```
MAKE ”X 1
SZAMITAS :X
```

Ez is ugyanazt az eredményt adná.

Ha a SZAMITAS eljárásban a gép minden értéket kiszámított és kinyomtatott, ismét a KAMATTABLAZAT nevű főprogram veszi át az irányítást és előhívja a következő, HOSSZABBITAS? nevű eljárást.

```
TO HOSSZABBITAS?
 IF :IDO > 15 THEN U 40 "- TABLAZATFEJ ADATFELIRAS
END
```

Ebben vizsgáljuk meg, hogy a táblázatfej nem csúszott-e ki a képrnyőről. (IF :IDO > 16 ...). Ha igen, úgy a gép először egy 40 karakter hosszúságú vonalat húz, majd a TABLAZATFEJ és az ADATFELIRAS eljárásokat hívja fordított sorrendben, hogy újra kiírják a táblázat fejlécét.

```
TO EREDMENY
 TAB 9 [VEGTOKE =] S 1 PRINT1 :TOKE S 1 PR [FT]
END
```

Az EREDMENY eljárással végül, a gép kiírja a végeredményt és befejezi a programot.

Új fogalmak a LOGO-ban:

első tapasztalatok a lokális és globális változókkal.

6.4 PASCAL PROGRAMOZÁS

A Pascalban is mellőzhetjük az alapprogramot és azonnal rátérhetünk a részletes kidolgozott program ismertetésére. Felépíthetnénk a programot ugyanazzal a szerkezettel is mint a BASIC-ben vagy a LOGO-ban, de jobbnak tartjuk, ha kiszélesítjük a megoldási lehetőségeket. Ennek következményeként az eljárások, illetve az alprogramok nem egészen ugyanazt fogják nyújtani.
Míg a LOGO-ban a gép tárában található összes eljárást minden programnál előhívhatjuk (valódi parancskiterjesztés!), ugyanez a Pascalban rendkívüli előkészületeket igényel. Ezért az 5. fejezet ide illő eljárásait nem vettük át változtatás nélkül, hanem némileg módosítottuk azokat. Így a feladatot az alábbi eljárásokkal oldhatjuk meg:

— KEPERNYOBEVITEL
— FEJ
— SZAMITAS

A FEJ eljárás két aleljárást tartalmaz: a TABLAZATFEJ és a BEVITELIKEP eljárásokat. Új dolog a FEJ eljárásban, hogy előhívásakor értéket adunk a változójának. A példánkban ezt az értéket a **fent** nevű változó kapja. Az eljárásfej mutatja meg az átadandó paraméter adattípusát:

PROCEDURE FEJ (fent :Boolean);

Ezután az eljárás átveszi a **fent** változó (jelen esetben logikai változó) értékét és dolgozhat vele. A **fent** változó az eljárásban lokális, vagyis csak ott ismert és csak ott lehet használni. Ennek a változónak az eljárás meghívásánál adunk értéket. Az átadandó érték az eljárás neve mögötti zárójelben található.

Mivel a feladatot részfeladatokra bontottuk, a főprogram lényegében a részfeladatokat megoldó eljárások hívásából áll:

— képernyőfelépítés;
— táblázatfej előállítása (FEJ (ture));
— számítás;
— a táblázat hosszúságának vizsgálata, szükség esetén a FEJ (false) eljárás előhívása;
— a megoldás kinyomtatása.

Mindezekkel a következő Pascal programot kapjuk:

```
program kamatszamitas_tablazattal;
{ A program adott kezdeti tokebol, kamatlabbol }
{ es a kamatozasi idobol szamitja ki a vegto-  }
{ ket. A kamatozasi idot csak egesz evekben    }
{ adhatjuk meg. A program a toket es a kama-   }
{ tokat minden evre tablazat formajaban mu-    }
{ tatja be.                                    }
var kezdeti_toke,toke,kamatlab,kamatok:real;
    kamatozasi_ido                    :integer;
```

```
procedure kepernyobevitel;
    begin
       clrscr;
       writeln(' K A M A T S Z A M I T A S ');
       writeln(' ======================== ');
       writeln;writeln;
       writeln(' A vegtoke kiszamitasa');writeln;
       write(' Kezdeti toke   Ft-ban?');readln(kezdeti_toke);
       write(' Kamatlab       %-ban?');readln(kamatlab);
       write(' Kamatozasi ido ev-ben?');readln(kamatozasi_ido);
       toke:=kezdeti_toke;
       kamatlab:=kamatlab/100;
    end;
procedure fej(fent:boolean);
{ Kiirja a tablazatfejet es a felhasznalando adatokat. }
{ A "fent" parameter hatarozza meg a fej formajat az-  }
{ szerint, hogy a tablazat elejen vagy a vegen van.    }
    procedure tablazatfej;
       begin
          writeln('     Ev        Evi        Ev    ');
          writeln('  kezdete     kamat     vege   ');
          writeln('   ( Ft )     ( Ft )    ( Ft ) ');
       end;
    procedure bevitelikep;
       begin
          writeln('================================');
          writeln('k=',kezdeti_toke:8:2,' Ft  p=',
                  kamatlab*100:3:1,' %  i=',kamatozasi_ido:3,' ev');
          writeln('================================');
       end;
    begin   { fej }
       if fent then
          begin
             clrscr;
             bevitelikep;
             tablazatfej;
             writeln('-------------------------------');
          end
       else
          begin
             writeln('-------------------------------');
             tablazatfej,
             bevitelikep;
          end;
    end;   { fej }
procedure szamitas;
var i:integer;
    begin
       for i:=1 to kamatozasi_ido do
          begin
             write(i:3,toke:8:2);
             kamatok:=toke*kamatlab;
             write(kamatok:8:2);
             toke:=toke+kamatok;
             writeln(toke:8:2);
          end;
    end;
```

```
begin {foprogram}
    kepernyobevitel;
    fej(true);
    szamitas;
    { Tablazatfejet kiiratni, ha eltunt a felso szelrol. }
    if kamatozasi_ido>16 then
       fej(false);
       { eredmeny }
    writeln('    A vegtoke      = ',toke:8:2,' Ft ');
end.
```

A KEPERNYOBEVITEL eljárás tökéletesen azonos a 5.4 alfejezet megfelelő eljárásával. A FEJ eljárás két aleljárását — a TABLAZATFEJ és az BEVITELIKEP eljárásokat — a **fent** változónak adott érték (true v. false) szerint két különböző sorrendben használjuk.

A FEJ eljárás során tapasztalhatja először, hogy egy eljáráson belül ismét eljárásokat definiálhatunk. Az eljárások ugyanúgy épülnek fel, mint a programok csak a PROCEDURE szóval kezdődnek. Ezzel a sorrendet is megadtuk: először a programban és az eljárásokban is a változókat deklaráljuk (ebben az esetben ilyen nincs), azután az aleljárásokat, és csak azután az eljárás utasítás részét. Milyen előnye van az eljárások efajta egymásba skatulyázásának? Ez egészen egyszerű: ha például az esetünkben a két aleljárást — TABLAZATFEJ és BEVITELIKEP — a FEJ eljáráson kívül deklaráljuk, akkor ez bárhonnan hívható lenne, vagyis a főprogramból is és a FEJ eljárás utasítás részéből is. Ha viszont az aleljárásokat a FEJ eljáráson belül deklaráljuk, akkor csak ezen az eljáráson belül használhatók, kívül egyáltalán nem ismertek, nem hívhatók. Ebben az esetben ez a különbség lényegtelen, sok más esetben azonban biztosabb és jobb, ha az eljárásokat csak ott deklaráljuk, ahol szükség van rájuk.

Lehet, hogy a FEJ eljárás utasítás része kissé meglepő. Az eddigi hasonló parancsokban az IF és a THEN között egy kifejezés állt, amely vagy igaz volt, vagy hamis. (pl. IF Kamatozasiido > 15 THEN ...). A programunkban ez a kifejezés maga a **fent** nevű változó (IF fent THEN ...). Azaz, ha **fent** értéke igaz (true), a THEN utáni utasítást hajtja végre a gép, de ha **fent** értéke hamis (false), akkor az ELSE utáni utasítássor végrehajtása következik.

A számítási eljárás is tartalmaz újdonságot: az eljárás belsejében deklarálunk egy változót. Mint ahogy az előbbiekben már megállapítottuk, az ilyen változó csak az eljáráson belül ismert, csak ott használható, az eljáráson kívül már nem. Ennek megfelelően előfordulhat a programban két változó is ugyanazzal a névvel: az egyik az eljáráson belül, a másik pedig ezen kívül. A mindenütt ismert, mindenütt használható változókat a program deklarációs részében kell megadni: ezeket *globális* változóknak nevezzük. Az **i** változót, amelyet a számítás eljáráson belül deklarálunk, csak ez az eljárás ismeri, ezért ezt *lokális* változónak nevezzük. (Analóg módon definálјuk a LOGO-ban a globális változókat a MAKE"... utasítással, míg a csak egyetlen eljárásban szereplő változók a lokális változók (l. 6.3 alfejezet). A BASIC-ben minden változó globális.)

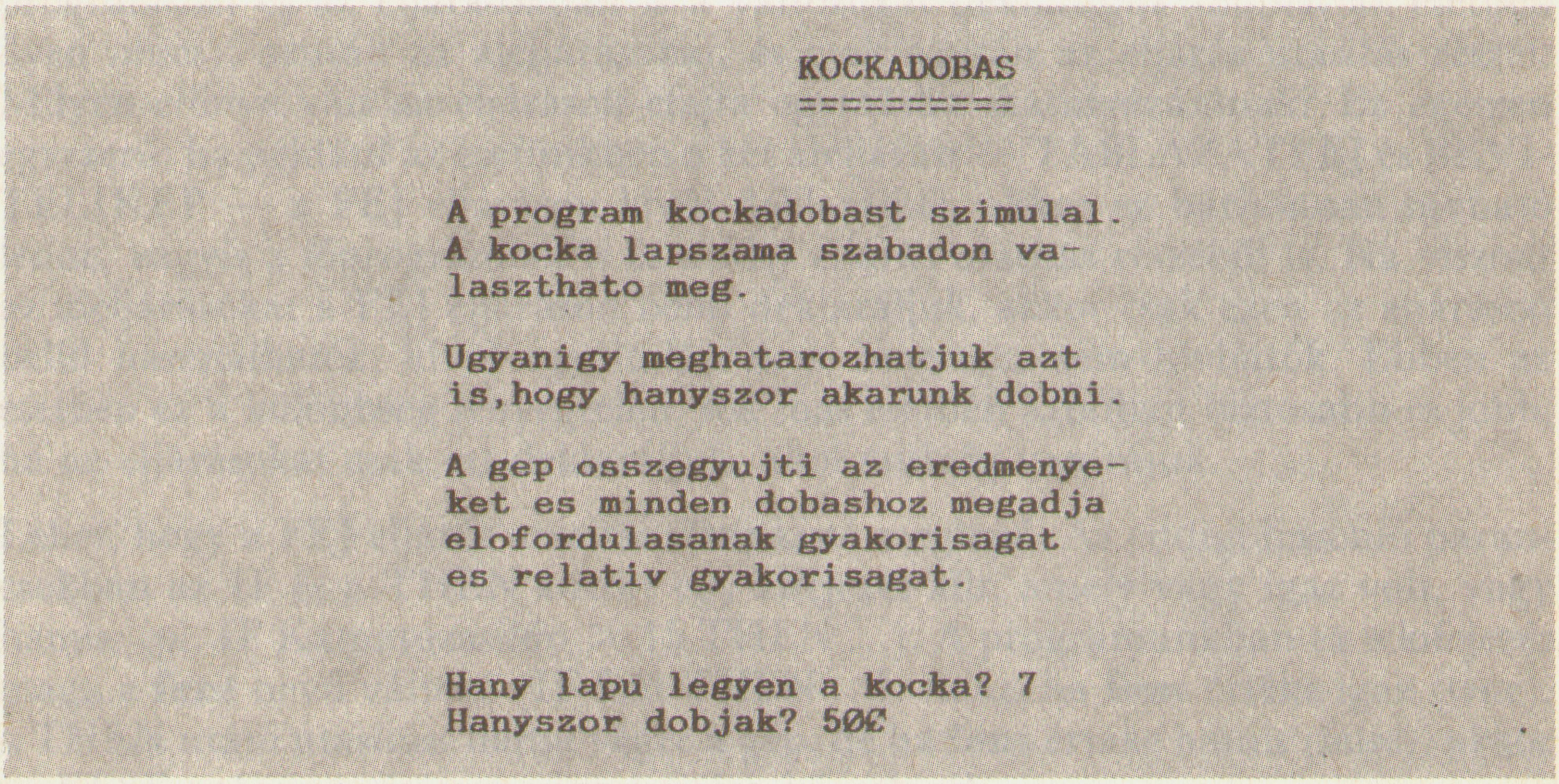
Az eljárás nyomtatási utasítása olyan jobbra illesztett kiírást eredményez, ahol az egymást követő sorokban a tizedespontok egymás alatt helyezkednek el. A 2. fejezetben már ismertettük a formátumra vonatkozó utasítást.

Új fogalmak a Pascalban:
obális és lokális változók, eljárások eljáráson belül.

7. HÉTOLDALÚ KOCKA?

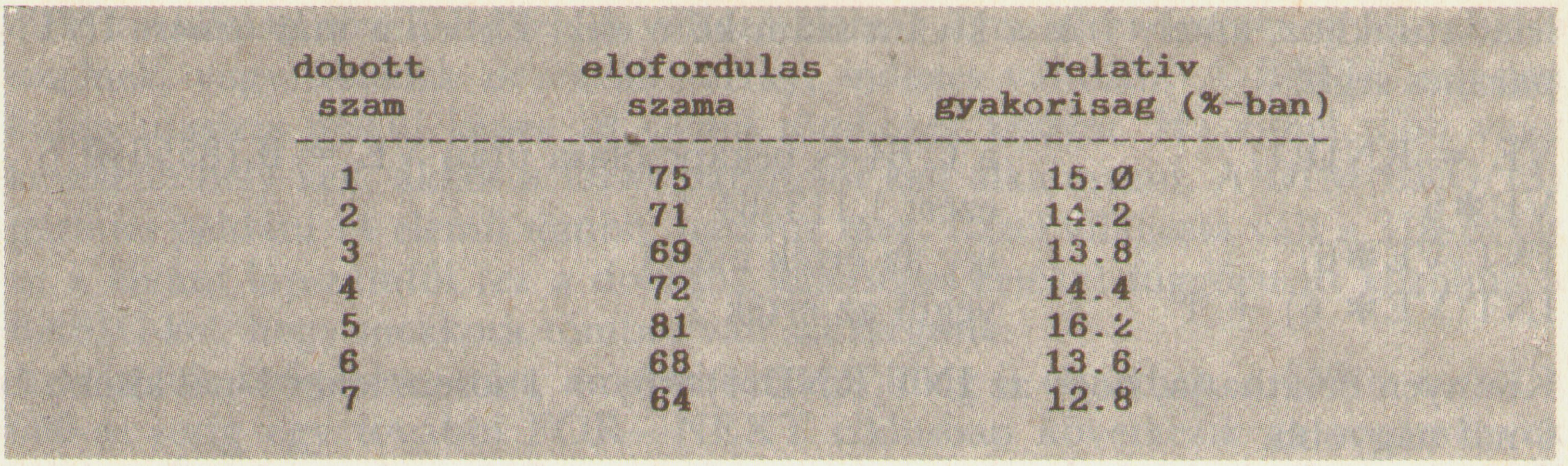
7.1 A FELADAT MEGHATÁROZÁSA

A programmal kockadobálást szimulálunk. A kocka lapajainak száma tetszőlegesen választható, Így szimulálhatjuk a szokványos hatlapú kockát, de ugyanígy bármilyen más „kockát” is tetszőleges lapszámmal. A feladatot megfogalmazhatnánk úgy is, hogy egy szerencsekereket szimuláljunk tetszőlegesen megadható *n* számú egyenlő méretű szektorral.



```
                    KOCKADOBAS
                    ==========

         A program kockadobast szimulal.
         A kocka lapszama szabadon va-
         laszthato meg.

         Ugyanigy meghatarozhatjuk azt
         is,hogy hanyszor akarunk dobni.

         A gep osszegyujti az eredmenye-
         ket es minden dobashoz megadja
         elofordulasanak gyakorisagat
         es relativ gyakorisagat.


         Hany lapu legyen a kocka? 7
         Hanyszor dobjak? 500
```

A második szám beírásával határozzuk meg a dobások számát. A végén statisztikát kapunk az egyes dobott számok gyakoriságáról. Ehhez táblázat formájában kell megadni a képernyőn a dobott számot, előfordulásának számát és azt, hogy ez a szám az összes dobás hány százaléka. Hogy a felhasználónak megkönnyítsük a munkát, mindjárt a program elején tájékoztatjuk, hogy mire képes, mit tud a program. Ezt mutatjuk be a következő ábrán:



```
  dobott           elofordulas              relativ
   szam               szama           gyakorisag (%-ban)
 ---------------------------------------------------------
    1                  75                  15.0
    2                  71                  14.2
    3                  69                  13.8
    4                  72                  14.4
    5                  81                  16.2
    6                  68                  13.6
    7                  64                  12.8
```

Szintén az első képernyőn kéri a gép a felhasználótól az oldallapok és dobások számának a megadását. Ha beírjuk a kívánt adatokat, a következő felirat jelenik meg:

Egy pillanat, dobok…

Ez a szöveg addig marad a képernyőn, amíg a gép végre nem hajtotta a dobásokat és nem közölte az egyes dobások előfordulásait és százalékos megoszlásukat. Az ezután következő képernyőn láthatjuk az eredményt.
Biztosan felmerül Önökben a kérdés, hogyan dobhat egy számítógép? Erre mindhárom programnyelven van egy-egy parancs, amely egy véletlenszámot ad. Bár ez a beépített *véletlenszám generátor* nem is ad mindig annyira véletlen számokat. Kipróbálhatja a véletlenszám generátorának minőségét, ha elég sokszor „dobat" a számítógéppel (10 000 vagy még több dobás). Ideális generátornál a különböző dobások annál egyenlőbb számban fordulnak elő, minél több a dobás, ami azt jelenti, hogy a relatív gyakoriságuk közeledik egymáshoz.
A véletlenszám generátorálás kezdőértékét meg tudjuk változtatni a RANDOM vagy RANDOMIZE utasítással. Azt, hogy az egyes nyelvekben ez pontosan hogyan megy végbe, megnézheti a megfelelő leírásban.

7.2 KIDOLGOZÁS BASIC-BEN

Először azt beszéljük meg, hogyan állíthatunk elő a BASIC-ben véletlenszámokat. Az ezt végrehajtó BASIC parancs az **RND(X)**. Sajnos, a BASIC interpreterek közötti hasonlóság az **X** argumentumnál be is fejeződik. Legjobb, ha a számítógép kézikönyvében utánanéz, hogy az Ön gépe milyen argumentumot kér. Egyes gépeknél az első RND() parancs előtt még a RANDOMIZE utasítást is be kell írni. Itt a COMMODORE C 64-es és az IBM BASIC szintaxisát ismertetjük. Általában az RND (1), egy 0 és 1 közé eső véletlenszámot ad a határok kizárásával. Ebből a számból azután egy olyan számot kell létrehozni a feladatunkhoz, amely 1 és a kívánt szám közé esik. Ebben a már ismert INT() parancs segít:

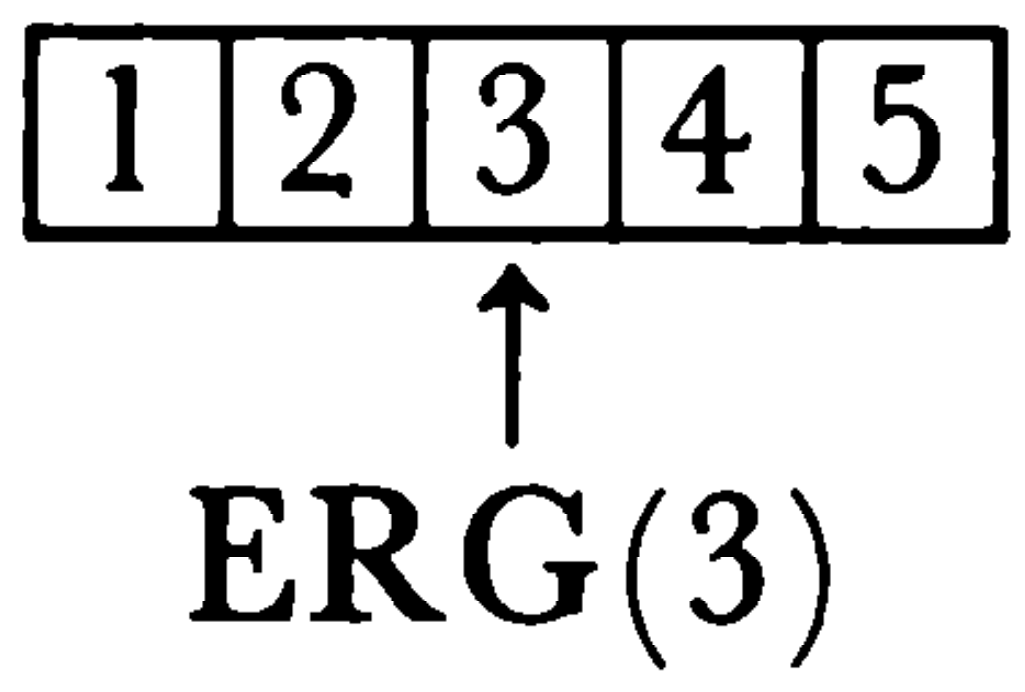
VE = RND(1)	a C 64-es bekapcsolása után VE = 0.185564016
VE*6	ezért 1.113384
INT(VE*6)	így az érték már 1
INT(VE* 6) + 1	végül az érték 2

Könnyen beláthatja, hogy az INT(RND(1)*6) + 1 sosem lehet 1-nél kisebb és 6-nál nagyobb.

Az alapprogram tehát:

```
250 INPUT FL
260 INPUT AZ
310 DIM ERG(FL)
320 FOR I=1 TO AZ
330 ZU=INT(RND(1)*FL+1)
340 ERG(ZU)=ERG(ZU)+1
350 NEXT I
650 FOR I=1 TO FL
660 PRINT I,ERG(I),
670 PRINT INT(ERG(I)/AZ*1000+0.5)/10
680 NEXT I
690 END

READY.
```

A 320-as sortól kezdődik a számítási rész. Egy FOR—NEXT ciklussal szimulálja és számolja a gép a dobásokat. A ciklus annyiszor fut le, amennyit **AZ** értékének megadtunk az INPUT részben (... I = 1 TO AZ). A 330-as sorban minden átmenetnél a gép a már megbeszélt módon létrehoz egy 1 és a választott lapszám (FL) közé eső véletlenszámot. Ez lesz ZU pillanatnyi értéke. A következő (340) sorban számlálja a dobásokat. Hogyan kell ezt a számlálást értelmeznünk? Tételezzük fel, hogy egy ötoldalú kockával dobunk. A számláláshoz tehát öt változóra van szükségünk, amelyek mindegyike az öt lehetséges dobás közül egynek az előfordulásit számlálja. Amikor éppen azt a számot dobtuk, a változó tartalma mindig 1-gyel növekszik. Célszerű mind az öt változónak ugyanazt a nevet adni. Programunkban ezt az ERG() rövidítéssel oldjuk meg. A zárójelben levő egész számok különböztetik meg egymástól az egyes változókat: ezek a számok egyfajta mutatót (indexet) jelentenek. A mutató mindig arra az elemre mutat, amelyik a dobott számnak felel meg. Az ábra azt az esetet szemlélteti, amikor a 3-as számot „dobtuk”.



Egymás mellett látja az öt elemet. A mutató a harmadik elemre mutat. A BASIC-ben ez a mutató egy szám.

Ha a FOR—NEXT ciklus valamelyik átmeneténél a 330-as sorban például 2-est dobunk, akkor **ZU** értéke 2 lesz. Az **ERG(ZU) = ERG(ZU) + 1** ekkor a 2-es elem tartalmát növeli 1-gyel. A változó elemek ilyen elrendezését egydimenziós *mező*nek, vagy *tömb*nek nevezzük (array, indexes változó). A BASIC minden külön utasítás nélkül képes 10 elemű tömböket kezelni. Ha ennél több vagy kevesebb hely kell, akkor a DIM parancsot kell alkalmazni. A DIM ERG(13) parancs például 14 elemű tömböt határoz meg. Ezzel világossá vált a 310-es sor is: a kívánt lapszámtól (ez a dobások lehetséges értéke) függően itt közöljük a BASIC-kel, hogy mekkora tömbre van szükségünk.

A számítás befejezése után, amely hosszabb ideig is eltarthat, a 650-es sortól a 680-as sorig egy további FOR—NEXT ciklusban következik az eredmények kiírása. Érdekes még a 670-es sor. Itt az egyes számok dobásainak a számából számítja ki a gép az adott szám relatív gyakoriságát a

$$\text{relatív gyakoriság } (\%) = \frac{\text{a szám előfordulása} * 100}{\text{összes dobás}}$$

képlettel. Az INT(rel.gyakoriság*10 + 0,5)/10 képlettel az eredményt egy tizedesjegyre kerekítjük.

Ha ezt az alapprogramot a szokásos módon kiegészítjük kommentárokkal és tájékoztatással a következő részletes programot kapjuk:

```
10 REM KOCKADOBAS
20 REM --------------------------------------------------
30 REM X VALASZTHATO LAPSZAMU KOCKAVAL TETSZOLEGES SOKSZOR
40 REM DOBUNK. A PROGRAM MINDEN DOBAST MEGSZAMOL ES
50 REM UTANA A DOBAS SZAZALEKOS GYAKORISAGAT,
60 REM 1...X-IG BEMUTATJA.
70 REM --------------------------------------------------
99 :
100 REM  PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"♥":REM CLS
120 PRINT TAB(11)"K O C K A D O B A S"
130 PRINT TAB(11)"==================="
140 PRINT:PRINT
150 PRINT"A PROGRAM KOCKADOBAST SZIMULAL."
160 PRINT"A KOCKA LAPSZAMAT SZABADON"
170 PRINT"VALASZTHATJUK MEG!"
175 PRINT
180 PRINT"UGYANIGY HATAROZHATJUK MEG, HOGY"
190 PRINT"HANYSZOR AKARUNK DOBNI."
195 PRINT
200 PRINT"A SZAMITOGEP OSSZEGYUJTI AZ EREDMENYE-"
210 PRINT"KET ES MINDEN DOBASHOZ MEGADJA AZ ELO-"
220 PRINT"FORDULASANAK GYAKORISAGAT ES RELATIV"
230 PRINT"GYAKORISAGAT SZAZALEKOS ERTEKBEN."
240 PRINT:PRINT:PRINT
250 INPUT"KEREM A LAPSZAMOT ";FL
260 INPUT"HANYSZOR DOBJAK   ";AZ
270 PRINT
280 PRINT"     EGY PILLANAT, DOBOK..."
299 :
300 REM 'AZ'-SZER DOBNI
310 DIM ERG(FL)
320 FOR I=1 TO AZ
330 ZU=INT(FL*RND(1)+1)
340 ERG(ZU)=ERG(ZU)+1
350 NEXT I
```

```
599 :
600 REM EREDMENY ES VEGE
610 PRINT"□":PRINT:   REM CLS
620 PRINT"DOBOTT     ELOFORDULAS       RELATIV GYA-";
630 PRINT"SZAM           SZAMA         KORISAG (%)"
640 PRINT"------------------------------------------"
650 FOR I=1 TO FL
660 PRINT TAB(2);I;TAB(15);ERG(I);TAB(30);
670 PRINT INT(ERG(I)/AZ*1000+.5)/10
680 NEXT I
690 END

READY.
```

Ez a program semmi újat sem tartalmaz az eddigiekhez képest, így a programból kiolvasható a program menete is.

Új fogalmak a BASIC-ben:

RND, RANDOMIZE, DIM, egyváltozós tömbök felépítése, indexes változók használata

7.3 MEGOLDÁS LOGO-BAN

A LOGO-ban a RANDOM paranccsal állítunk elő véletlenszámokat. A RANDOM utasításhoz még egy kiegészítő adatot is meg kell adni. Ezzel azt a tartományt határozzuk meg, ahová a véletlenszám esik. A RANDOM :N például a 0 és N—1 közé eső véletlen egész számokat állít elő a határokat is beleértve. Mivel a kocka számai 1-gyel kezdődnek, 1-et hozzáadunk. Így az 1 + RANDOM 5, 1 és 6 közé eső véletlen egészeket állít elő. A most következő alapprogramocska százat dob, és mind a száz dobást kiírja:

```
TO KOCKADOBAS :N
 DOBNI 1
END
```

A KOCKADOBÁS főeljárást a
KOCKADOBÁS 7

begépelésével hívhatjuk elő, ha egy 7 lapú kockával akarunk dobálni.

```
TO DOBNI :L
 IF :L > 100 THEN STOP
 PRINT ( 1 + RANDOM :N ) PRINT "''
 DOBNI :L + 1
END
```

A DOBNI eljárásban az utolsó utasításkor (DOBNI :L + 1) újra előhívja az eljárást, de a változó (:L) értékét 1-gyel megnövelni. Ez addig tart, amíg a változó értéke el nem éri a százat. Ha ez bekövetkezik, akkor a STOP utasítással térünk vissza a főprogramba és ezzel a program futása be is fejeződik.

Ha a DOBNI eljárásból elhagyjuk az **IF :L > 100 THEN STOP** megszakítási feltételt és az utolsó sort (DOBNI :L + 1), akkor csak egy dobás lesz. A részletes műkocka programot a következő eljárásokra osztjuk:

```
TO MUKOCKA
 TAJEKOZTATAS
 ADATBEADAS
 ELOKESZITES 1
 DOBAS 1
 EREDMENY 1
END
```

A TAJEKOZTATAS eljárás írja ki a képernyőre a használati utasítást.

```
TO TAJEKOZTATAS
 ND TAB 14 [K O C K A D O B A S] CR 1
 TAB 14 "= U 19 "= CR 3
 PRINT [A PROGRAM KOCKADOBALAST MODELLEZ]
 PRINT [A 'KOCKALAPOK' SZAMA SZABADON VALASZT-]
 PRINT [HATO,UGYANIGY A DOBASOK SZAMA IS]
 PRINT [A SZAMITOGEP OSSZEGZI AZ EREDMENYT ES]
 PRINT [MEGADJA AZ EGYES DOBASOK ABSZOLUT ES]
 PRINT [RELATIV GYAKORISAGAT.] CR 3
END
```

Az ADATBEADAS eljárás (ugyanazon a képernyőn) kéri a felhasználótól a szükséges adatokat.

```
TO ADATBEADAS
 PRINT [HANY LAPJA] S 3 PRINT1 [LEGYEN A 'KOCKANAK'] S 1
 MAKE "LAPSZAM INPUT
 PRINT [HANYSZOR DOBJAK?] S 1 MAKE "DSZAM INPUT
 CR 2 TAB 6 [PILLANAT,DOBOK...]
END
```

A LOGO nyelvben nincs a tömbnek (DIM vagy ARRAY) megfelelő alapszó, mint a BASIC-ben és a Pascalban. Ezen úgy segíthetnénk, hogy minden dobási eredményhez hozzárendelünk egy változót, amely számlálóként működik. A megfelelő változó értéke a dobott számok alapján (IF ... THEN ...) 1-gyel növekedhetne. Azt azonban előre kellene tudni, hogy hányféle dobás fordulhat elő; ez a módszer tehát nem alkalmas a feladat megoldására. Ezért a LOGO egy olyan tulajdonságát használjuk ki, amellyel sem a BASIC, sem a Pascal nem rendelkezik; a programban annyi változót állítunk elő, amennyire szükségünk van. Ezt az ELOKESZITES eljárással tesszük, amely a kocka minden lapjához egy változót rendel nulla kezdetőértékkel.

```
TO ELOKESZITES :G
 IF :G > :LAPSZAM THEN STOP
 MAKE ( WORD "W :G ) 0
 ELOKESZITES :G + 1
END
```

Az így hozzárendelt változók első jele mindig a W betű, második jelük pedig egy szám, amely 1-től a kocka lapjainak számáig bármilyen egész szám lehet. Minden szám természetesen csak egyszer fordulhat elő. Az imént említett változókat a **WORD ”W :G** utasítással állítjuk elő. A WORD parancs az utána következő két adatot egy szóvá fűzi össze: példánkban a **W** betűt **:G** értékével. Mivel **:G** az 1, 2, ..., kocka lapszám értékeket veszi fel egymás után, a változók a W1, W2, ... neveket kapják. Ezeket a „nekveket” a MAKE parancs már változókként értelmezi, mindegyiknek nulla értéket rendel.

```
TO DOBAS :G
 IF :G > :DSZAM THEN STOP
 MAKE "E ( RANDOM :LAPSZAM ) + 1
 MAKE ( WORD "W :E ) ( 1 + THING ( WORD "W :E ) )
 DOBAS :G + 1
END
```

A DOBÁS eljárás 1-től a kocka lapszámáig állít elő véletlenszámokat és az **E** változóban helyezi el (MAKE ”E (RANDOM :LAPSZÁM) + 1). Az **:E** értéke határozza meg a következő programsorban azt a változót, amelynek az értéke 1-gyel fog növekedni. A **THING (WORD ”W :E)** úgy adja meg a **WORD ”W :E** változó értékét, hogy ezt az értékét **1 + ...**-vel megváltoztathassuk. Ez az eljárás addig hívja magát folyamatosan — mindig eggyel növelve **:G** értékét —, amíg **:G** értéke el nem éri a kívánt dobások számát (IF :G > DSZAM THEN STOP).

```
TO EREDMENY :L
 TABLAZATFEJ
 KIIRAS :L
END
```

Az EREDMENY eljárás két eljáráshívást tartalmaz:

```
TO TABLAZATFEJ
 ND CR 1
 PRINT1 [DOBOTT] TAB 13 [DOBASOK] TAB 32 [%-OS] CR 1
 PRINT1 [SZAM] TAB 13 [SZAMA] TAB 32 [ARANY] CR 1
 U 35 "- CR 1
END
```

A TABLAZATFEJ állítja elő a képernyőn a táblázat fejlécét,

```
TO KIIRAS :L
 IF :L > :LAPSZAM THEN CR 2 STOP
 MAKE "H THING ( WORD "W :L )
 MAKE "R ( ROUND 1000 * :H / :DSZAM ) / 10
 TAB 3 :L TAB 16 :H TAB 31 :R CR 1
 KIIRAS :L + 1
END
```

a kiírás pedig kinyomtatja az egyes dobások számát (ezek a :H változóban vannak) és relatív gyakoriságát (ezek pedig :R változóban találhatók). Ez utóbbit ugyanazzal a képlettel számítjuk ki, mint a BASIC-ben és egy tizedesre kerekítjük.

A 12. fejezetben szerepel majd két olyan eljárás (FELVAG és APRIT), amelyekkel ugyanez a probléma másképp is megoldható.

Új fogalmak a LOGO-ban:

WORD összetett szavak létrehozása (ebben a példában változók nevei), THING változónévnek értékadás.

7.4 PASCAL PROGRAMOZÁS

Először itt is a véletlenszámok előállításával foglalkozunk. Ehhez a standard Pascalban nincs beépített függvény, de mint már említettük, az általunk tárgyalt mindhárom Pascalban van.
Vannak azonban olyan matematikai képletek, amelyek többé-kevésbé véletlenszámokat állítanak elő. Ezek közül mutatunk be egy egyszerű függvényt, amely a kívánt tartományba (1 — kockalapszám) eső véletlenszámokat ad eredményül:

```
program fuggveny;
var lapszam , veletlen_uj,i  : integer;
function veletlen :integer;
begin
   veletlen_uj := veletlen_uj*32763 ;
   veletlen:=abs(trunc(veletlen_uj/32763*lapszam))+1
end;
begin
   write('Kerem a lapszamot:');readln(lapszam);
   for i:=1 to 100 do
   writeln(veletlen)
end.
```

A veletlen_uj nevű változót a főprogram deklarációs részében egész típusú változóként kell definiálni, és a veletlen elnevezésű függvény első hívása előtt adunk neki egy tetszőleges értéket. Így a program bármely helyén az

a := veletlen;

értékadással véletlenszámokat állíthatunk elő.

A TURBO—Pascalban ezekre az előkészületekre nincs szükség, mert ott van beépített véletlenszám függvény. Ennek ellenére megírjuk a veletlen_szam nevű függvényt, amelyben a RANDOM függvényt használjuk. Ennek megfelelően választhat az itt bemutatott függvény és a Pascaljának véletlenszám függvénye között.
A TURBO—Pascalban is, éppúgy mint a LOGO-ban, a RANDOM(*n*) értéke 0 és *n*—1 közé esik, beleértve a határokat is. A következő program egy véletlenszámot hoz létre 1 és a választott kockalapszám között:

```
program kockadobas;
var lapszam      :integer;
function veletlen_szam :integer;
begin
    veletlen_szam:=random(lapszam)+1
end;
begin
    readln(lapszam);
    writeln(veletlen_szam)
end.
```

Ha 50 véletlenszámot akar, egészítse ki az utasítás részt egy megfelelő FOR—TO—DO ciklussal.

Ebben a programban találkozik először ún. saját függvénnyel, amit Ön írt meg. Használtuk már a feladatmegoldások során a Pascal beépített függvényeit, pl. a ROUND függvényt (kerekítés). A függvényeket olyan eljárásoknak tekintjük, amelyeknek egyetlen eredményparaméterük van. Ugyanúgy építjük fel őket, mint az eljárásokat. Az első sor a FUNCTION szóval kezdődik, ezt követi a függvény neve és a függvény eredményének adattípusa. A függvényértéket a függvény nevéhez rendeljük, ezért az utasítás részben szerepelni fog egy értékadó utasítás, amelyben a függvény nevével egy változóhoz rendeljük a függvény értékét. Ha meg akarjuk számolni a dobásokat, mint ahogy ezt a feladat kéri, a számláláshoz változó kell. Ehhez egy egydimenziós tömböt használunk, amelynek az adattípusa ARRAY. A BASIC-kel ellentétben, ennek a tömbnek az indexe nem feltétlen 1-gyel (vagy egyes BASIC nyelvjárásokban 0-val) kezdődik. A Pascalban meg kell adnunk a kezdeti és a végső indexet is, amivel rögtön meg is határoztuk a tömb elemeinek a számát is. Amint ezt a Pascalban már megszoktuk, meg kell még adni a tömbben tárolt adatok típusát is. Mivel a legtöbb Pascal változatban az egész típusú változók értéke csak −32 767 és +32 767 között lehet, valós típusút választunk, hogy igen sok „dobás" legyen lehetséges. Egyébként megfelelne az egész típus is.
A Pascalban nemcsak változókat, hanem konstansokat is lehet alkalmazni. Ezek az elemek csak „olvashatók", értéküket a deklarációs részben rögzítjük. Itt a **vege** konstanst használjuk, értéke 20. Ha ennél több lapú kockát kívánunk, akkor az állandónak más nagyobb értéket kell adni.

Ezek voltak a részletes program legfőbb nehézségei. Lássuk a programot:

```
program kockadobas_n_lapszamu_kockaval;
        {Megvalaszthato n lapszamu kockaval}
        {tetszoleges sokszor dobunk.Minden }
        {dobast megszamolunk es ezutan be- }
        {mutatjuk az 1...n dobasok relativ }
        {gyakorisagat.                     }
const vege=20;  {A lapok maximalis szama.   }
var   eszamlalo          :array [1..vege] of real;  {elofordulasok szama}
      dszam,mutato       :real;   {dobasok szama}
      szazalek           :real;
      lapszam            :1..vege;
      dobszam            :integer;  {dobott szam}
```

```
function vel           :integer; {veletlenszam}
  begin
    vel:=random(lapszam)+1
  end;
procedure kepernyo;
  begin
    clrscr;
      writeln('               KOCKADOBAS');
      writeln('               ==========');
      writeln;writeln;
      writeln('A program kockadobast szimulal.');
      writeln('A kocka lapszama szabadon va-  ');
      writeln('laszthato meg.');
      writeln;
      writeln('Ugyanigy meghatarozhatjuk azt');
      writeln('is,hogy hanyszor akarunk dobni.');
      writeln;
      writeln('A gep osszegyujti az eredmenye-');
      writeln('ket es minden dobashoz megadja');
      writeln('elofordulasanak gyakorisagat ');
      writeln('es relativ gyakorisagat.   ');
      writeln;writeln;
      write('Hany lapu legyen a kocka? ');
      readln(lapszam);
      write('Hanyszor dobjak? ');
      readln(dszam);
      writeln;writeln('   Egy pillanat, dobok...');
      writeln;writeln;
    end;
procedure dobni;
var i:integer;
  begin {az eszamlalo[ ]-t megtolteni nullakkal}
    for i:=1 to vege do
      eszamlalo[i]:=0;
      {dobni es megszamolni}
      mutato:=1;
      repeat
        dobszam:=vel;
        eszamlalo[dobszam]:=eszamlalo[dobszam]+1;
        mutato:=mutato+1;
       until mutato>dszam;
  end;
procedure eredmeny;
var i:integer;
  begin
    clrscr;
    writeln('dobott       elofordulas          relativ');
    writeln(' szam          szama        gyakorisag (%-ban)');
    writeln('----------------------------------------------');
    for i:=1 to lapszam do
      begin
        szazalek:=round(eszamlalo[i]/dszam*1000)/10;
        writeln(i:3,eszamlalo[i]:15:0,szazalek:20:1)
      end
  end;
  begin {F O P R O G R A M}
    kepernyo;
    dobni;
    eredmeny
  end.
```

A deklarációs részben a következő sorrendet kell betartanunk:

— állandók;
— változók;
— függvények.

A véletlenszám függvényt már használtuk az alapprogramban.
A KEPERNYO eljárás, mint általában, tájékoztatja a felhasználót és átveszi a futtatáshoz szükséges adatokat a felhasználótól. Ezek után a számítógép számol. Amíg a számítás tart, a képernyőn az

Egy pillanat, dobok...

felirat látható.
Magát a dobást a gép a DOBNI eljárás során hajtja végre. Először az eredményszámláló tömböt tölti fel nullákkal. A következő REPEAT—UNTIL ciklusban addig dob a kockával, amíg el nem éri a megadott dobásszámot. A **dobott_szam := vel_szam** utasítással a vel_szam nevű függvényt hívja elő, és az azzal előállított véletlenszámot a **dobott_szam**-nak adja értékül. Az eredményszámláló tömbben pedig 1-gyel nő a dobott számnak megfelelő helyen álló szám. Ennek a ciklusnak a végén tehát az összes dobások száma eloszlik az eredményszámláló tömb egyes rekeszeiben.
Az EREDMENY eljárás először a táblázat fejlécét nyomtatja ki. Ezután egy FOR—TO—DO ciklussal kiszámítja a dobások lehetséges értékeihez tartozó gyakoriságot, az összes dobások százalékában (számítási képlet a BASIC részben), egy tizedesre kerekíti, majd kiíratja a dobott számmal és az abszolút gyakorisággal egy sorba.
A főprogram csak a három eljárást hívja a megfelelő sorrendben.

Új fogalmak a Pascalban:

RANDOM beépített függvény, konstans és tömb definiálása (ARRAY), saját függvények definiálása és hívása.

8. DÉLOSZTÓL A JELENKORIG — KÖZELÍTŐ ÉRTÉKEK SZÁMÍTÁSA

8.1 A FELADAT MEGHATÁROZÁSA

Azt mutatjuk be egy példán, hogyan lehet intervallumok egymásba skatulyázásával közelítő értéket kiszámítani.

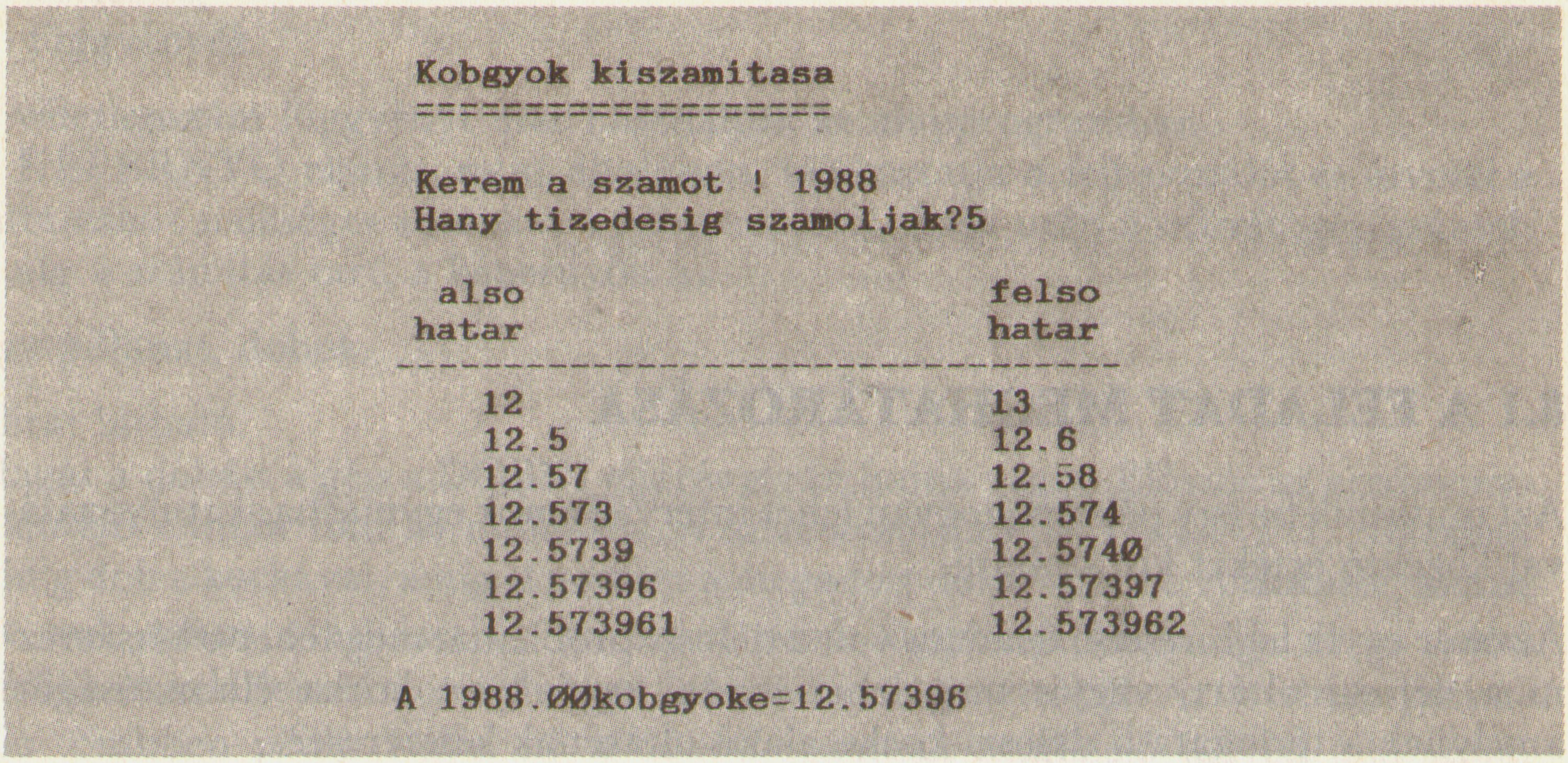
Az ókor egyik leghíresebb feladata volt a déloszi probléma: meg lehet-e kétszerezni a déloszi oltárt, vagyis meg lehet-e adni egy olyan kocka élhosszúságát, amelynek a térfogata a déloszi kocka alakú oltárénak kétszerese?
Akkoriban igen sok filozófiai elmélet foglalkozott ezzel a témával, de matematikai megoldás nem született.
A feladat 2 köbgyökének a kiszámításával oldható meg. A harmadik gyök kiszámítására írunk programot az egymásba skatulyázott intervallumok módszerével. Ennek a módszernek az az előnye, hogy más függvények közelítő értékeinek kiszámításához is könnyen alkalmazható, ellentétben a matematikailag elegánsabb módszerekkel. A mi igen egyszerű eljárásunkban mindig a 0 kezdeti értékkel indítjuk a folyamatot. Ezt a kezdeti értéket addig növeljük ugyanazzal a D növekménnyel, amíg a harmadik hatványa meg nem haladja a gyök alatti mennyiséget. Ekkor egy lépést visszamenve kapjuk az első közelítést. Ezután D-t eredeti értékének tizedére csökkentjük és megismételjük az eljárást. Minden ilyen lépéssorozatnál elméletileg egy új tizedesjegyet kapunk. A gyakorlatban azonban a BASIC és a LOGO legelterjedtebb változatainál csakhamar nehézségekbe ütközünk a kerekítési hibák miatt.

A módszert 2 harmadik gyökének kiszámításán mutatjuk be:

0 * 0 * 0 < 2	növekmény 1
1 * 1 * 1 < 2	növekmény 1
2 * 2 * 2 > 2	tehát vissza, és megvan az első közelítés
1,1 * 1,1 * 1,1 < 2	most a növekmény 0,1
1,2 * 1,2 * 1,2 < 2	növekmény 0,1
1,3 * 1,3 * 1,3 > 2	tehát vissza, és megvan a második közelítés
1,21 * 1,21 * 1,21 < 2	most a növekmény 0,01
1,22 * 1,22 * 1,22 < 2	növekmény 0,01

A felhasználónak kell eldöntenie, hány tizedes pontossággal kíván számolni.

A következő képernyőábra mutatja, milyen tájékoztatást kell nyújtania a programnak:



```
 Kobgyok kiszamitasa
 ===================

 Kerem a szamot ! 1988
 Hany tizedesig szamoljak?5

  also                     felso
 hatar                     hatar
-----------------------------------
    12                     13
    12.5                   12.6
    12.57                  12.58
    12.573                 12.574
    12.5739                12.5740
    12.57396               12.57397
    12.573961              12.573962

A 1988.00kobgyoke=12.57396
```

A számítások korlátozott pontossága a BASIC-ben és a LOGO-ban ahhoz vezethet, hogy a kerekítési hibák miatt a közelítés nem egyenletes. Így ebben a két nyelvben — esetenként — áttekinthetetlen képernyőkiírást kapunk.

Sok BASIC változatban dupla pontossággal is számolhatunk. Ehhez a változókat általában a # jellel látjuk el. A LOGO különféle változatai is eltérnek egymástól a pontosság tekintetében.

FILE:EA211 M: 4-3 PTC4-3 G Basic

8.2 KIDOLGOZÁS BASIC-BEN

Az alapprogramban először a **ZA** gyökalapot és a **GN** pontosságot kell megadnunk:

```
310 UG=0:S=1

READY.
```

A 8.1 alfejezetben leírtaknak megfelelően az **UG** alsó határ és az **S** lépéstávolság kezdeti értékeit rögzítjük:

```
160 INPUT ZA,GN

READY.
```

A felső és alsó határok kiszámítását a hozzá tartozó ellenőrzéssel egy FOR—NEXT ciklusba tesszük:

```
340 FOR I=1 TO GN+2
360 UG=UG+S
370 IF UG*UG*UG<=ZA THEN 360
380 OG=UG:UG=UG-S
400 PRINT UG,OG
410 S=S/10
420 NEXT I

READY.
```

A harmadik gyök ellenőrzési feltétele a 370-es sorban áll. Ha más függvényt akarunk kiszámítani, akkor az **UG*UG*UG < = ZA** feltételt megfelelőképpen át kell alakítani. Amíg a feltétel nem teljesül, **UG** még nem az alsó határ és a 360-as sorban értéke ismét **S**-sel növekedik. Ha azonban, a feltétel teljesül, megtaláltuk a pillanatnyi lépéstávolsághoz tartozó felső határt (OG = UG). Az alsó határ ennél egy lépéstávolsággal kisebb (UG = UG — S).
Ezután a lépéstávolságot tizedrészére csökkentjük (410-es sor), és addig ismételjük a folyamatot, amíg a kívánt pontosságot el nem érjük. Ahhoz, hogy az utolsó számjegy pontosságát biztosítsuk, legalább még egy jegyet kell kiszámítani, majd törölni. A ciklus **GN+2**-ig tart. A BASIC-ben és a LOGO-ban előfordulhat, hogy már az első tizedesjegy is bizonytalan lesz a kerekítési hibák miatt. Ajánlatos ezért a számítást kétszeres pontossággal elvégezni (#), ha van ilyen.

```
10 REM EGY SZAM KOBGYOKE
20 REM -------------------------------------------
30 REM A PROGRAM EGY TETSZOLEGES SZAM KOBGYOKET
40 REM SZAMITJA KI INTERVALLUMOK EGYMASBASKATULYA-
50 REM ZASANAK MODSZEREVEL. AZ EREDMENY LEHETSEGES
60 REM PONTOSSAGA CSAK AZ INTERPRETER MUVELETI
70 REM PONTOSSAGATOL FUGG.
80 REM -------------------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"♥":REM CLS
120 PRINT"        KOBGYOK KISZAMITASA "
130 PRINT"        =================== "
140 PRINT
150 PRINT"MELYIK SZAMNAK SZAMITSAM KI "
160 INPUT"A KOBGYOKET         ";ZA
165 INPUT"HANY TIZEDESJEGYRE  ";GN
170 PRINT
180 PRINT"  ALSO                    FELSO"
190 PRINT" HATAR                    HATAR"
200 PRINT"--------------------------------"
299 :
300 REM AZ INTERVALLUMOK KISZAMITASA
310 UG=0 :  REM ALSO HATAR
330 S=1  :  REM LEPESKOZ
340 FOR I=0 TO GN+1
360 UG=UG+S
370 IF UG*UG*UG<=ZA THEN 360
380 OG=UG
390 UG=UG-S
400 PRINT TAB(3);UG;TAB(26);OG
410 S=S/10
420 NEXT I
599 :
600 REM EREDMENY ES VEGE
```

```
610 PRINT
620 A$=STR$(OG)
630 A$=LEFT$(A$,LEN(A$)-1) :REM AZ UTOLSO JEGY TUL SOK
640 PRINT"            KOBGYOK(";ZA;")=";A$
650 :
660 END

READY.
```

A részletes megoldást kommentárral kezdjük azoknak, akik később foglalkozni akarnak még a programmal. A felhasználónak szánt információkat most is, mint eddig, a 100-astól a 299-es sorokig helyeztük el. Ugyanitt, ebben a programrészben kérdezzük meg az INPUT paranccsal a gyökalap és a pontosság értékeit.
A számítási rész teljesen azonos az alapprograméval. A program jobb érthetősége kedvéért bizonyos lépéseket különválasztottunk. Egy-egy részszámítás befejeztével a gép a megtalált határokat balra igazítva kinyomtatja (400-as sor).
A számítás befejezése után még egyszer tájékoztatást adunk az elért eredményről.

Amint azt már elmondtuk, fontos, hogy a számítást egy tizedesjeggyel tovább végezzük a kívántnál, és az utolsó jegyet törölve kapjuk az eredményt a kívánt pontossággal. Ahhoz, hogy a BASIC-ben az utolsó helyről egy számot töröljünk, először a 620-as sorban a számértéket karakterlánccá alakítjuk (A$= STR$(OG)). Így felhasználhatjuk a karakterláncokra vonatkozó műveleteket. Az utolsó jegyet a LEFT$(A$,IG) függvénnyel vágjuk le. A függvény első argumentuma (A$) az a karakterfüzér, amin a műveletet végezzük, a második (IG) pedig a balról levágott eredményfüzér hossza. Mivel esetünkben csupán egyetlen karaktert akarunk elhagyni, IG értéke a teljes füzérhossznál (LEN(A$)) eggyel kisebb (630-as sor). (A karakterfüzéreken végzett műveletek közül a RIGHT$(A$,IG) a jobbról vágást, míg a MID$(A$, TOL, DB) a füzér belsejéből való kivágást teszi lehetővé.)

A felhasznált BASIC változattól függően a kerekítési hibák már korán kiderülhetnek. Ez esetben nem kapunk olyan áttekinthető képernyőkiírást, mint a képernyőábránkon. Tehát, az ilyen esetben nem kell programhibára gyanakodnunk. Esetleg végtelen ciklust is kaphatunk.

Új fogalmak a BASIC-ben:

STR$ (), LEFT$ (), LEN ().

8.3 MEGOLDÁS LOGO-BAN

Az alapprogramot nevezzük KOZ (közelítés)-nek. Az alábbi paraméterek kellenek:

— **:N** a pillanatnyi alsó határ;
— **:D** a mindenkori lépéstávolság;

— **:X** a már kiszámított tizedesjegyek száma;
— **:R** a szám, amelynek a köbgyökét számoljuk;
— **:G** a kívánt tizedesjegyek száma.

Ezekkel a jelölésekkel a következő alapprogramot kapjuk:

```
TO KOZ :N :D :X :R :G
 IF :X > :G THEN TOPLEVEL
 TEST :N * :N * :N > :R
 IFT ( PRINT ( :N - :D ) :N ) KOZ ( :N - :D ) ( 0.1 * :D ) ( :X + 1 ) :R :G
 IFF KOZ ( :N + :D ) :D :X :R :G
END
```

A **TEST :N * :N * :N > :R** feltétel biztosítja, hogy a harmadik gyököt számítsuk ki. Ha az ellenőrzési feltétel teljesül (IFT), kiíratjuk a kapott közelítő értékeket ((PRINT(:N — :D):N)), és ismét meghívjuk a KOZ eljárást az új alsó határral ((:N — :D)), a tizedére csökkentett lépéstávolsággal (0.1 * :D) és **:X** értékét 1-gyel megnövelve. A **:X** lépésszámláló mindig megmutatja, hogy hányadik tizedesnél tartunk, ezért növeljük meg az értékét minden átmenetnél 1-gyel. Az első programsorban fogalmaztuk meg a leállási feltételt (IF :X > :G). Ha a vizsgált feltétel (TEST ...) nem teljesül (IFF), akkor is újra meghívjuk az eljárást a **:D** lépéstávolsággal megnövelt közelítő értékkel (:N + :D), de nem változtatjuk meg sem a **:D** lépéstávolságot, sem az **:X** számlálót. Ha az eljárást ellenőrizni akarjuk, az eljárás neve mögött értékeket kell adnunk minden lokális változónak.

KOZ 0 1 0 8 4

például a 8 harmadik gyökét számítja ki 4 tizedesjegyre. Egy ötparaméteres eljárás meglehetősen áttekinthetetlen. Ha két eljárásra bontjuk, leegyszerűsödik az alapprogram.

```
TO GYOKVON :R :G
 KOZH 0 1 0
END

TO KOZH :N :D :X
 IF :X > :G THEN STOP
 TEST :N * :N * :N > :R
 IFT ( PRINT ( :N - :D ) :N ) KOZH ( :N - :D ) ( :D / 10 ) ( :X + 1 )
 IFF KOZH ( :N + :D ) :D :X
END
```

Most csak két, a számítással összefüggő paraméterértéket, a gyök alatti mennyiséget és a pontosságot kell beírni:

GYOKVON 8 4

A részletes program neve GYOK, és GYOKKEZDES, a KOZELITES és az EREDMENY eljárásokból áll. Szükségünk van még a KOZELITES eljáráson belül a KIIRAS nevű eljárásra is, hogy az egymásba skatulyázott intervallumok közbülső értékeit formáltan írathassuk ki. A KOZ eljárás öt lokális változója közül kettőt globális változóval helyettesítünk, mint a KOZH eljárásban. Ezeket a felhasználó a GYOKKEZDES eljárás futása során írhatja be. Ezzel a kis változássl vesszük át a KOZ alapeljárást a részletes programba. A többi eljárás nem tartalmaz semmi újat. Így a részletes program minden további magyarázat nélkül is érthető.

```
TO GYOK
 GYOKKEZDES
 KOZELITES 0 1 0
 EREDMENY
END

TO GYOKKEZDES
 ND CR 1
 TAB 4 [A 3.GYOK] S 2 PRINT1 [KISZAMITASA] CR 1
 TAB 4 "= U 21 "= CR 2
 PRINT [MELYIK SZAM HARMADIK GYOKET]
 TAB 14 "'' PRINT [SZAMOLJAM KI?] S 1
 MAKE "R INPUT
 PRINT [HANY TIZEDESJEGYRE ?] S 1
 MAKE "G INPUT
 CR 1 TAB 4 [ALSO] TAB 27 [FELSO]
 CR 1 TAB 4 [HATAR] TAB 27 [HATAR] CR 1
 U 35 "- CR 1
END

TO KOZELITES :N :D :X
 IF :X > :G + 1 THEN MAKE "ERG :N STOP
 TEST :N * :N * :N > :R
 IFT KIIRAS KOZELITES ( :N - :D ) ( 0.1 * :D ) ( :X + 1 )
 IFF KOZELITES ( :N + :D ) :D :X
END

TO KIIRAS
 TAB 4 ( :N - :D ) TAB 27 :N CR 1 U 35 "- CR 1
END

TO EREDMENY
 CR 1
 PRINT1 [A SZAM (] PRINT1 :R PRINT1 [) HARMADIK GYOKE =] S 1
 PRINT1 BL :ERG CR 2
END
```

Az elérhető pontosság attól függ, hogy milyen pontossággal végzi el a gép a szorzásokat. A C 64-es LOGO öt tizedesjegyet vesz figyelembe, ezért ezzel a módszerrel — a harmadik gyököket is — legfeljebb négy tizedesjegyig tudunk számolni. A kerekítési hibák miatt nem lesz olyan jól áttekinthető a képernyő, hogy a minden lépésben javított közelítést pusztán ránézéssel is észlelhessük.
A nyomtatás ráadásul balra igazítottan történik, hiszen arról már régebben lemondtunk, hogy ezen változtassunk. Léteznek azonban olyan LOGO változatok is, amelyek 1000 tizedesjegyre adják meg az eredményt (IBM—LOGO).

Új fogalom a LOGO-ban:

BUTLAST (BL).

8.4 PASCAL PROGRAMOZÁS

A Pascalban a megoldás elsősorban a számítás eljárásában rejlik. Ezért tekintsük azonnal a teljes programot:

```
program kobgyokszamitas;
  { A program tetszoleges szam kobgyoket szamitja ki  }
  { egymasba skatulyazott intervallumok hasznalataval.}
  { Az eredmeny pontossaga csak a compilertol fugg.    }
var alsohatar,felsohatar,gyokalap,
   lepestavolsag,eredmeny         :real;
   tizedesjegyek,tizedesjegy      :integer;
procedure kepernyobevitel;
   begin
     clrscr;
     writeln(' Kobgyok kiszamitasa ');
     writeln(' =================== ');writeln;
     write(' Kerem a szamot ! ');readln(gyokalap);
     write(' Hany tizedesig szamoljak?');readln(tizedesjegyek);
     writeln;
     writeln('  also                      felso');
     writeln(' hatar                      hatar');
     writeln('-------------------------------');
   end;
procedure szamitas;  var tizedesjegy :integer;
   begin
     alsohatar:=0;lepestavolsag:=1;
     for tizedesjegy:=0 to tizedesjegyek+1 do
          begin
          repeat     {felsohatar meghatarozasa}
          alsohatar:=alsohatar+lepestavolsag;
          until alsohatar*alsohatar*alsohatar>gyokalap;
          felsohatar:=alsohatar;
          alsohatar:=alsohatar-lepestavolsag;
          writeln('    ',alsohatar:1:tizedesjegy,
                        felsohatar:23:tizedesjegy);
          lepestavolsag:=lepestavolsag/10;

          end;
    end;
 begin { foprogram }
    kepernyobevitel;
    szamitas;
    eredmeny:=alsohatar;writeln;
    writeln('A ',gyokalap:1:2,'kobgyoke=',eredmeny:1:tizedesjegyek);
 end.
```

A **gyokalap** és a **tizedesjegyek** nevű változók — értéküket a felhasználó határozza meg — mellett a program kidolgozásához további változók is kellenek: egy-egy a határoknak (alsó határ, felső határ), a lépésköznek és az eredménynek. A KEPERNYOBEVITEL eljárás eddigi példáink azonos nevű eljárásai szerint épül fel, ezért nem is kell tovább magyarázni.

A SZAMITAS eljárásban először az **alsohatar** és a **lepestavolsag** kezdeti értékét adjuk meg (0, illetve 1). A szűkebb értelemben vett számítást két egymásba ágyazott ciklussal oldjuk meg. A külső ciklust, amely egy FOR—TO—DO ciklus, annyiszor futtatjuk le, ahány tizedesjegyre akarunk számolni. A belső ciklusban addig növeljük az alsó határt, amíg a harmadik hatványa meg nem haladja a gyökalapot (REPEAT ... UNTIL alsohatar * alsohatar * alsohatar > gyokalap). Ezt követően előkészítjük a következő tizedesjegy kiszámítását. Ehhez a **felsohatar** az **alsohatar** pillanatnyi értékét kapja, az **alsohatar** pedig, a lépésközzel csökken. Ezután kiíratjuk az így kapott két értéket, majd a **lepestavolsag** a tizedére csökken. Ismét a külső ciklusba kerülünk, újra indul a folyamat, s tart a következő tizedesjegy meghatározásáig.

A főprogramban egymás után hívjuk meg a KEPERNYOBEVITEL és a SZAMITAS eljárást. Az utolsó **alsohatar** adja meg az eredményt. Eggyel több tizedesjegye van, mint a **tizedesjegyek** változóban rögzített érték, mert a pontosság érdekében egy lépéssel tovább számoltunk. A már többször alkalmazott elegáns formázási utasítással (... eredmeny : 1 : tizedesjegyek) az eredményt a kívánt számú tizedesjeggyel íratjuk ki.

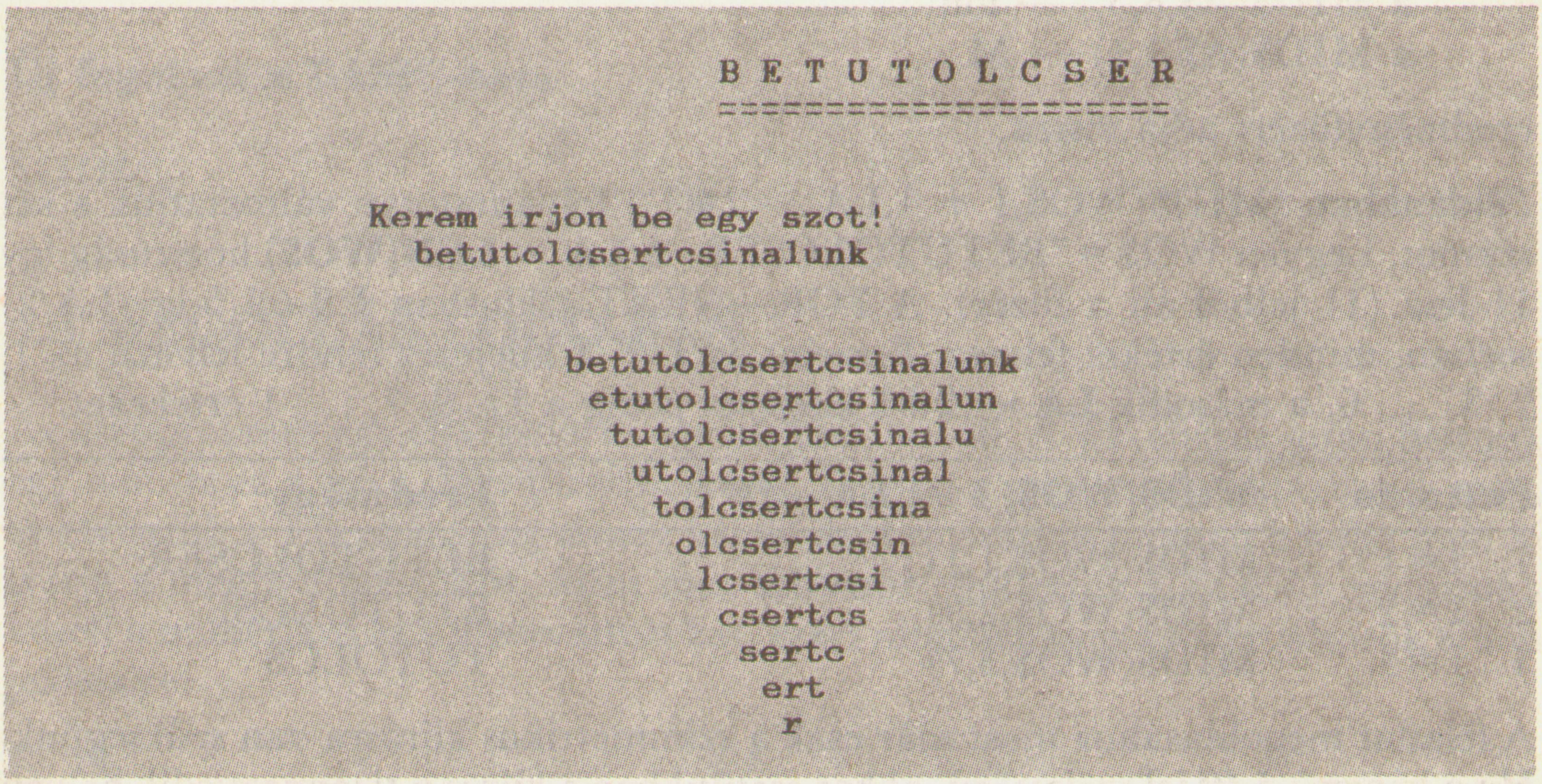
Új fogalmak a Pascalban:

változó használata a megformázási utasításban.

9. BETŰTÖLCSÉR

9.1 A FELADAT MEGHATÁROZÁSA

A számítógép egy füzér (string) beírását kéri. A füzért a képernyő közepére szimmetrikusan írja ki úgy, hogy sorról sorra haladva a füzér első és tulsó jelét mindig vágja le. A képernyőábra a program egy lehetséges futását mutatja be:



```
                B E T U T O L C S E R
                =====================


Kerem irjon be egy szot!
  betutolcsertcsinalunk


         betutolcsertcsinalunk
          etutolcsertcsinalun
           tutolcsertcsinalu
            utolcsertcsinal
             tolcsertcsina
              olcsertcsin
               lcsertcsi
                csertcs
                 sertc
                  ert
                   r
```

9.2 KIDOLGOZÁS BASIC-BEN

Az alapprogram bemenő adatként egy tetszőleges füzért vár, amit aztán az előírás szerint fog feldolgozni.
Nézzük hogyan:

```
160 INPUT WO$
310 LA=LEN(WO$)
320 X=(40-LA)/2
330 FOR I=0 TO LA/2
340 A$=MID$(WO$,I+1,LA-2*I)
350 PRINT TAB(X+I);A$
360 NEXT I

READY.
```

Ezt az alapprogramot később a főprogramban, lényegében ugyanígy találjuk meg.

Ha egy változóhoz egy füzért rendelünk hozzá, akkor a változó nevét $ jellel kell lezárni (WO$), különben hibajelzést kapunk.
A LEN(WO$) függvény a füzér hosszát adja meg, vagyis, hogy hány karakterből áll a füzér.
A MID$ függvény szintén füzérekre alkalmazható, de egészen más jellegű művelet, mint az előző. Lehetővé teszi, hogy egy füzérből tetszőleges helytől indulva tetszőleges hosszú darabot kiragadjunk, kivágjunk. MID$(A$,TOL,DB,) — ez azt jelenti, hogy a zárójelen belül három dolgot kell megadni. Az első (A$) maga a füzér, a második (TOL) egy szám, amely azt a helyet adja meg, ahonnan indulva a második szám (DB) által meghatározott hosszúságú darabot ki akarjuk vágni.
Legyen A$ = "Magnetofon",

ekkor a MID$(A$,4,4,) — neto
a MID$(A$,8,3,) — fon
a MID$(A$,1,1) — M

eredményfüzért adja vissza.

Példánkban (MID$(WO$,I + 1,LA — 2*I)) folyamatosan változtatjuk a két számot. Legyen WO$ = "BETUTOLCSER. A **LA = LEN(WO$)** hosszúság így 11 lesz (11 jelből áll a füzér). A FOR—NEXT ciklusban **I** 0-tól 5-ig (LA/2) sorban, minden értéket felvesz. Eszerint az **I + 1** kifejezés rendre felveszi az 1, 2, 3, ..., 6, a második **LA — 2*I** kifejezés pedig a 11, 9, 7, ..., 1 értékeket.

Futtatás	**MID$(WO$, I+1,LA—2*I)**	**Eredmény**
1 I=1	MID$(WO$,1,11)	BETUTOLCSER
2 I=2	MID$(WO$,2,9)	ETUTOLCSE
3 I=3	MID$(WO$,3,7)	TUTOLCS

A feladat megoldásához most már csak a szimmetrikus kiírásra van szükségünk. Ha a képernyő 40 jel szélességű, akkor esetünkben a kiírás a tizennegyedik helyen kezdődik ((40 — 11)/2). Ezzel a 320-as sort is megmagyaráztuk.
Ahhoz, hogy a ciklus minden átfuttatásánál az egyre rövidülő füzért mindig egy hellyel jobbra kezdjük írni, a tabulátort (TAB(X + I)) mindig növelni kell eggyel. A főprogram az alapprogramból, némi kommentárral és a tájékoztató résszel kiegészítve alakul ki:

```
10 REM BETUTOLCSER
20 REM ----------------------------------------------
30 REM TETSZOLEGES SZOT TOLCSER FORMABAN IRUNK KI UGY,
40 REM HOGY AZ ELEJEN ES A VEGEN MINDIG EGY-EGY BETUT
50 REM LEVAGUNK ES AZ IGY KAPOTT SZOT AZ
60 REM ELOZO SZO ALA IRJUK.
70 REM ----------------------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
```

```
110 PRINT"□": REM CLS
120 PRINT"        B E T U T O L C S E R"
130 PRINT"        ====================="
140 PRINT:PRINT
150 PRINT"KEREM IRJON BE EGY SZOT!"
160 INPUT WO$
170 PRINT:PRINT
299 :
300 REM LEVAGAS ES KIIRATAS
310 LA=LEN(WO$) :REM A SZO HOSSZA
320 X=(40-LA)/2 :REM A KIIRAS KEZDETENEK HELYE
330 FOR I=0 TO LA/2
340 A$=MID$(WO$,I+1,LA-2*I)
350 PRINT TAB(X+I) A$
360 NEXT I
599 :
600 REM VEGE
610 PRINT:PRINT
620 END

READY.
```

Új fogalom a BASIC-ben:

MID$().

9.3 MEGOLDÁS LOGO-BAN

Az alapprogram egy eljárásból és egy aleljárásból áll. A TE eljárás során beírunk egy szót, és ez a szó a **:W** változóba kerül. Ezután a szó hosszát határozzuk meg, és átvisszük a TD eljárásba, amely kétoldalról rövidíti és minden alkalommal ki is írja a megrövidített szót. Mint már többször említettük a LOGO egyformán kezeli a numerikus és a füzérváltozókat, ezért nincs szükség a füzérváltozók külön jelölésére.

```
TO TE
 MAKE "W FIRST RQ
 TD COUNT :W
END


TO TD :X
 TAB ( 20 - :X / 2 ) :W CR 1
 TEST :X > 1
 IFF STOP
 IFT MAKE "W ( BF BL :W ) TD :X - 2
END
```

Mit csinál a TD eljárás? Az első sorral a már bevitt füzért (szó) nyomtatjuk ki úgy, hogy a képernyőn a szélektől egyenlő távolságra, szimmetrikusan helyezkedjen el. Ezután megvizsgáljuk, hogy a füzér hossza nagyobb-e, mint 1. Ha ez nem teljesül (IFF), akkor visszamegyünk a TE programba és a program futása ezzel befejeződik.

Ha a feltétel teljesül (IFT), akkor a BUTFIRST (BF) és a BUTLAST (BL) parancsokkal a szó első és utolsó karakterét levágjuk. A parancsok sorrendje itt nem számít. Az így kapott új **:W** füzérrel és a 2-vel csökkentett szóhosszúsággal újra előhívjuk a TD eljárást egészen addig, amíg a megszűnési feltétel ellenőrzése hamis eredményt ad. A COUNT LOGO-paranccsal egy lista elemeinek a számát határozzuk meg. Ez a parancs a legtöbb LOGO változatban nem alapszó, ezért írunk helyette egy eljárást (a BASIC-beli LEN-nek felel meg).

```
TO LEN :W
 IF :W = [] THEN OP 0
 OP 1 + LEN BF :W
END
```

A **:W** változóba azt a szót írjuk, amelynek a hosszúságát meg akarjuk állapítani. Ha nem írunk be egyetlen jelet sem ([]), az eljárás **0**-t ad eredményül, egyébként önmagát hívja, de levágja a szó első jelét és az OUTPUT (OP) értékét 1-gyel megnöveli. Ha már semmi sem maradt a szóból, a LOGO az OUTPUT értékét hátulról számítja ki, az üres szótól indulva. Ennek a speciális programozási lehetőségnek további felhasználásával még a következő fejezetben is foglalkozunk (l. még az A. függelék).

A részletes program nem sokban különbözik az alapprogramtól. A TOLCSERKEZDES eljárás a felhasználót tájékoztatja, a TOLCSERKIIRAS lényegében ugyanaz, mint a TD eljárás. A két eljárást a TOLCSER eljárással hívjuk elő.

```
TO TOLCSER
 TOLCSERKEZDES
 TOLCSERKIIRAS COUNT :W
END

TO TOLCSERKEZDES
 ND
 TAB 8 [B E T U T O L C S E R] CR 1
 TAB 8 "= U 20 "= CR 3
 PRINT [KEREM IRJON BE EGY SZOT!] CR 1
 TAB 3 ": MAKE "W INPUT CR 2
END

TO TOLCSERKIIRAS :X
 TAB ( 20 - :X / 2 ) :W CR 1
 TEST :X > 1
 IFF STOP
 IFT MAKE "W ( BF BL :W ) TOLCSERKIIRAS :X - 2
END
```

Látjuk, hogy a TOLCSERKIIRAS paramétere ismét függvénnyé válhat.

(Megjegyzés: mivel a LOGO a PRINTER parancs kiadása esetén mind a képernyőre, mind a nyomtatóra ír, vagy a képernyőn, vagy a nyomtatón lesz jó a nyomtatási kép. Próbálja meg úgy módosítani a programot, hogy a nyomtatón kapjon szép nyomtatási képet!)

Új fogalmak a LOGO-ban:

füzérek feldolgozása, BUTFIRST.

9.4 PASCAL PROGRAMOZÁS

A Pascalban is, mint a LOGO-ban, a részletes program tulajdonképpen úgy jön létre, hogy az alapprogramhoz hozzáírunk még egy eljárást, amivel a képernyőt építjük fel. Ezért nem foglalkozunk az alapprogrammal, hanem rögtön a részletes programot beszéljük meg.

A programot úgy alakítottuk ki, hogy könnyen át lehessen írni olyan képernyőre is, ahol egy sorba nem csak 40 karakter írható. Ezért vezetünk be egy állandót — **kepernyoszelesseg** —, amelynek az értéke programunkban 40. Ha a számítógépe egy sorba nem ennyi jelet ír, akkor ide a megfelelő értéket kell helyettesítenie.

Az állandókat olyan tárolóknak kell elképzelni, amelyeknek csak egyszer adhatunk értéket, éspedig akkor, amikor definiáljuk őket. Ettől kezdve a tartalmuk rögzített, nem adhatunk nekik más értéket a program futása során. Ha mégis megkísérelnénk új érték adását, akkor hibajelzést kapunk, és a program leáll. Az állandók deklarálását a CONST szóval kezdjük.
Ezt követi az állandó neve, az egyenlőségjel, majd az állandó értéke, végül mint általában, a pontosvessző zárja a sort. Természetesen több állandót is deklarálhatunk egyszerre, de a CONST szó csak egyszer szerepelhet, a legelején. Az állandók deklarációja mindig megelőzi a változók deklarációját.

Most pedig, ejtsünk szót a STRING nevű változótípusról. Amíg a CHAR adattípus csak azt adja meg, hogy karakteres változóról van szó, addig az új adattípus, a STRING, viszont lehetővé teszi a füzérek feldolgozását is, ugyanúgy, mint a BASIC-ben. Ezt az adattípust nem minden Pascal ismeri, de a ZOOM—Pascal és a TURBO—Pascal igen. A STRING adattípusú változó deklarálásakor a STRING szó után szögletes zárójelben fel kell tüntetni, hogy legfeljebb milyen hosszú lehet a füzér.

```
program betutolcser;
  { Tetszoleges szot tolcser formaban irunk ki ugy, }
  { hogy az elejen es a vegen mindig egy-egy betut  }
  { levagunk es az igy kapott szot az elozo ala ir- }
  { juk.                                            }
const kepernyoszelesseg=40;
  { Jel a kepernyo egy soraban. }
var szo               :string [kepernyoszelesseg];
procedure kepernyo;
  begin
     clrscr;
     writeln('B E T U T O L C S E R':(kepernyoszelesseg+35) div 2);
     writeln('=====================':(kepernyoszelesseg+35) div 2);
     writeln;writeln;
     writeln('Kerem irjon be egy szot!');
     write('  ');readln(szo);
     writeln;writeln;
  end;
procedure bemutatas;
var  szo_rovidites    :string [kepernyoszelesseg];
     szohosszusag,i   :integer;
     iraskezdet       :integer;
```

```
begin
  szohosszusag:=length(szo);
  iraskezdet:=szohosszusag+(kepernyoszelesseg-szohosszusag) div 2;
  for i:=0 to (szohosszusag div 2) do
     begin
        szo_rovidites:=copy(szo,i+1,szohosszusag-2*i);
        writeln(szo_rovidites :iraskezdet-i)
     end
end;
begin  { Foprogram }
  kepernyo;
  bemutatas
end.
```

Az első eljárás (KEPERNYO) informálja a felhasználót és beolvassa a feldolgozandó füzért.

A feldolgozást a BEMUTATAS eljárás végzi el: a füzér hosszát a LENGTH Pascal szóval állapítjuk meg és ezt az értéket adjuk a **szohosszusag** nevű változónak. A szimmetrikus kiíráshoz szükséges kezdeti kurzorpozíciót a **szohosszusag**ból és a **kepernyoszelesseg**ből számítjuk ki ugyanazzal a módszerrel, mint a BASIC-ben. A tölcsér egyes sorai egy FOR—TO—DO ciklusban alakulnak ki és rögtön meg is jelennek a képernyőn. A szóeleji és szóvégi jelek levágását a COPY függvény végzi, amelybe a füzért, a másolás kezdetét és a másolandó füzér hosszúságát kell beírnunk argumentumként. A COPY tehát megfelel a BASIC MID$-nak. Az így létrehozott lerövidített szót ezután a gép újra a képernyő közepére szimmetrikusan írja ki. A kiíráshoz szükséges kurzorpozíciót ugyanúgy számítjuk ki, mint a BASIC programnál.
A BEMUTATAS eljárásban találkozunk először az eljáráson belüli változó deklarálással. Ez szintén a VAR szóval kezdődik és deklaráljuk a

szó_rovidites : string ([kepernyoszelesseg]);
szohosszusag,i: integer;
iraskezdet : integer;

változókat, ahol az **i** a FOR—TO—DO ciklus futóindexe.
A különbség az, hogy ezeket a változókat az eljárás deklarációs részében definiáltuk, nem pedig a főprogram hasonló részében. Általánosságban erről azt mondhatjuk, hogy az eljáráson belül deklarált változókat csak ebben az eljárásban, illetve az ennek alárendelt eljárásokban használhatjuk.
A fölérendelt eljárásokban és természetesen a tulajdonképpeni programban ezek a változók ismeretlenek, nem hivatkozhatunk rájuk. Ennek az az előnye, hogy ugyanazt a változónevet más eljárásban egész más céllal is használhatjuk. Az ilyen változókat nevezzük *lokális* változóknak. Ez az esetünkben annyit jelent, hogy a **szo_rovidites, szo_hosszusag, iraskezdet** és **i** (futóindex) változók a BEMUTATAS eljárás lokális változói. Nem hahsználhatjuk tehát ezeket a főprogramban. A minden eljárás által használható változókat nevezzük globális változóknak.

A KEPERNYO és a BEMUTATAS eljárások alkotják az igen rövid főprogramot. Az ugyanis csak a két eljárás egymás utáni meghívásából áll.

Szeretnénk még röviden szólni egy eddig még meg nem magyarázott parancsról: a DIV osztási parancsról. Ez a parancs két egész számot oszt és egész eredményt ad:

99 DIV 6 eredménye 16

Az ebből az osztásból származó 3 maradékot a 4. fejezetben tárgyalt 99 MOD 6 művelettel kaphatjuk meg.

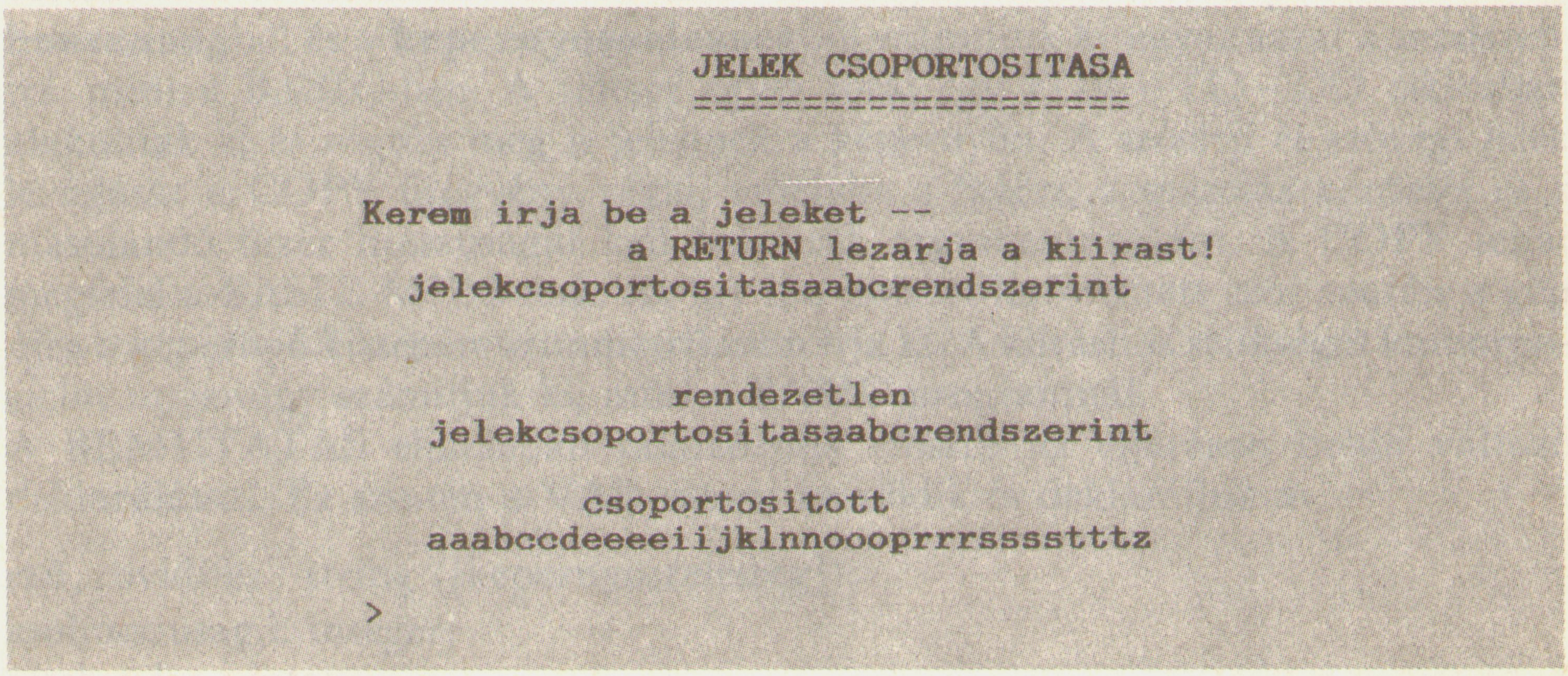
Új fogalmak a Pascalban:

CONST, LENGTH(), COPY(), lokális változók.

10. RENDEZÉS

10.1 A FELADAT MEGHATÁROZÁSA

A program egy, a billentyűzetről bevitt, tetszőleges karaktersorozat jeleit (betűk, számok, egyéb különleges jelek) válogatja szét és írja ki alfabetikus sorrendben. A feladat megoldása más, rendezési feladatoknál is felhasználható. A képernyőábra bemutatja, hogy milyen megoldást képzeltünk el:



```
                    JELEK CSOPORTOSITASA
                    ====================


         Kerem irja be a jeleket --
                a RETURN lezarja a kiirast!
           jelekcsoportositasaabcrendszerint


                      rendezetlen
            jelekcsoportositasaabcrendszerint

                  csoportositott
            aaabccdeeeeiijklnnoooprrrsssstttz

         >
```

A számítógép jelei kódolt jelek (rendszerint ASCII). Kézenfekvő, hogy a rendezés módszere: először a különleges jelek, azután a nagybetűk, majd a kisbetűk sorrendbe állítása legyen. Rendezési eljárás persze annyi van, mint égen a csillag, agyafúrtak és egyszerűek egyaránt. Van köztük nagyon gyors, van olyan is, amihez csak rövid program kell. Megint másokat speciális célokra terveztek, például nagyon nagy vagy nagyon kicsi adathalmaz rendezésére. Röviden: minden feladathoz van megfelelő módszer. Mi itt nem akarunk Önnek rafinált rendezési eljárásokat bemutatni, hanem egy alapjában véve nehézkes eljárást hasonlítunk össze az agyafúrtakkal. Előnye viszont, hogy könnyen áttekinthető, amit sok másikról nem mondhatunk el.

Képzelje el, hogy a következő számokat szeretné nagyság szerint rendezni:

3 6 1 7 8 2 5 4 9

Az általunk választott rendezési módszer — amelyet a BASIC és a Pascal megoldásban használunk — átnézi az egész számsort balról jobbra. Eközben mindig összehasonlítja az egymás melletti számokat, és ha például az első szám nagyobb, mint a második, akkor felcseréli a két számot. Ezután a második számot hasonlítja össze a harmadikkal stb. Az első átmenet példánkban a következő számsort adja:

3 1 6 7 2 5 4 8 9

Ebben a példában 8 számpár van. A program sorban összehasonlítja őket. Figyeljük meg a 8-as szám útját. Láthatjuk, hogy egészen hátrakerült a sorban. Az ötödik számpár (8,2) összehasonlításakor felcserélődött a 2-vel, a következő lépésben a (8,5) számpárnál az 5-tel cserélődik, majd a 4-gyel. A második átmenet végén a következő sorrend alakul ki:

1 3 6 2 5 4 7 8 9

Ennél a módszernél az első menetben a legnagyobb szám kerül az utolsó helyre. A második menetben a második legnagyobb szám kerül az utolsó előtti helyre stb. A módszer nem túl gyors, de igen áttekinthető.

A LOGO-ban alkalmazott módszer még egyszerűbb. Az első menetben kiválasztjuk a legnagyobb számot, és félretesszük. A többivel megismételjük a folyamatot, a végén pedig a félretett jeleket egyszerűen ebben a sorrendben visszaírjuk.

Megjegyzés: a LOGO-ban és az IBM BASIC-ben a 10 karakternél hosszabb, csak számokból álló karaktersorozat nem rendezhető ezzel a módszerrel, mert lebegőpontos számmá alakítják át.

10.2 KIDOLGOZÁS BASIC-BEN

A megoldásban eltekintünk az alapprogram ismertetésétől, mert megegyezik az utasítás résszel, ami a rendezés szempontjából a legfontosabb.

```
10 REM JELEK RENDEZESE
20 REM ---------------------------------
30 REM A PROGRAM EGY BEIRT 'SZOT' RENDEZ,
40 REM A JELEKET LEXIKOGRAFIAILAG RENDEZI.
50 REM ---------------------------------
99 :
100 REM PROGRAMFEJLEC ES ADATBEVITEL
110 PRINT"□" :   REM CLS
120 PRINT"         JELEK RENDEZESE"
130 PRINT"         ==============="
140 PRINT:PRINT
150 PRINT"KEREM IRJA BE A JELEKET-"
160 PRINT"A 'RETURN' LEZARJA A BEIRAST."
170 INPUT ZA$
299 :
300 REM A BEIRT FUZERT KULONALLO JELEKRE
310 REM BONTANI ES EZEKET AZ
320 REM FD$( ) MEZOBE VINNI.
330 LA=LEN(ZA$)
```

```
340 DIM FD$(LA-1)
350 FOR I=0 TO LA-1
360 FD$(I)=MID$(ZA$, I+1, 1)
370 NEXT I
380 REM AZ EREDETI FD$( ) MEZO KIIRATASA
390 PRINT:PRINT
400 PRINT TAB(15)"RENDEZETLEN"
410 GOSUB 1000 : REM FD$( ) MEZOT KIIRATNI
420 REM FD$( ) MEZO RENDEZESE
430 F=0 : REM SEGEDVALTOZO, AZT MUTATJA,HOGY RENDEZTUNK-E
440 FOR I=0 TO LA-2
450 IF FD$(I+1)>=FD$(I) THEN 500
460 H$=FD$(I)          :REM ELTESSZUK
470 FD$(I)=FD$(I+1)    :REM CSERELUNK
480 FD$(I+1)=H$
490 F=1 : REM VOLT CSERE
500 NEXT I
510 IF F=1 THEN 430 : REM TEHAT MEG NINCS KESZEN
599 :
600 REM EREDMENY ES VEGE
610 PRINT
620 PRINT TAB(16)"RENDEZETT"
630 GOSUB 1000 : REM FD$( ) MEZO KIIRATASA
640 END
996 :
997 :
998 REM ***************ALPROGRAM**********
999 :
1000 REM *** AP:FD$( ) MEZO KIIRATASA *****
1010 PRINT TAB((40-LA)/2);
1020 FOR I=0 TO LA-1
1030 PRINT FD$(I);
1040 NEXT I
1050 PRINT
1060 RETURN

READY.
```

A 99-edik sorig helyezzük el a kommentárokat, mint azt eddig is tettük.

A 100-as és a 299-es sorok között alakítjuk ki a képernyőt, és itt helyezzük el az adatbevitelt is (170 INPUT ZA$). A rendezendő füzér tehát a **ZA$** füzérváltozóba kerül.

A rendezés szempontjából sokkal szerencsésebb, ha a füzért különálló jelekre bontjuk. Először is tehát átalakítjuk a beírt „szót” különálló jelekre, amelyeket a 7. fejezetből már ismert tömbbe helyezünk el. Ez a 300-as és a 370-es sor közötti rész. Esetünkben ezt a tömböt **FD$()**-nak nevezzük, de azt is tudnunk kellene, hogy hány elemű legyen ez a tömb. Ezt a 330-as sorban határozzuk meg (a beírt füzér hossza), s a következő sorban már definiálhatjuk a tömböt (DIM FD$ (LA)). Az ezt követő FOR—NEXT ciklussal töltjük fel a tömb egyes helyeit (350—370-es sor).

A 360-as sorban a már ismert MID$() függvénnyel a **ZA$**-ból minden jelet az **FD$()** tömb elemeibe juttatunk, egymás után. Az elemeket a ciklusszámláló (I) „sorszámozza” 0-tól a hosszúság — 1-ig. Miért nem 1-től kezdjük és a hosszúságig haladunk? Nos, ezt is megtehettük volna. Ekkor viszont az **FD$()** tömb első rekeszébe nem írnánk semmit (az index 0-tól indul ki).

A program a rendezést a 420-as és az 510-es sorok között végzi. A 10.1-ben bemutatott módszert alkalmazzuk. Tudnunk kell azonban azt is, hogy a BASIC-ben nemcsak számokat, hanem jeleket is lehet nagyság szerint rendezni, éspedig a növekvő ASCII kódjuknak megfelelő sorrendben. A 450-es sorban mindig a két szomszédos elemet hasonlítjuk össze. **FD$(I)** jelenti a bal oldali elemet, **FD$(I + 1)** pedig a jobb oldalit. Ha a sorrend már helyes, akkor a következő párt hasonlítjuk össze. Ha még nem, akkor felcseréljük a két jelet. Az értékátadáshoz egy segédváltozót használunk (H$):

FD$(I) értéke **H$**-ba
FD$(I + 1) értéke **FD$(I)**-be
H$** értéke **FD$(I + 1)-be

Ezzel felcseréltük FD$(I) és FD$(I + 1) tartalmát. A FOR—NEXT ciklust addig kell futtatnunk, amíg ki nem javítottuk az összes "hibás" helyet. Hogy ezt ellenőrizhessük, az **F** változót használjuk jelzőként. Értékét a ciklus előtti 340-es sorban nullára állítjuk. Ha a cikluson belül átrendezés történt, akkor a 490-es sorban **F** értéke 1-re változik, és emiatt (510-es sor) újból lefut a ciklus. Ha nem volt újabb átrendezés (F = 0), akkor a rendezés befejeződik.
Már csak a kiírás hiányzik. Ezt egy alprogrammal végezzük el, amelyet már egyszer a 410-es sorban használtunk, amikor az eredeti füzért — karaktereiből ismét összeállítva — írattuk ki.
Most, hogy már legyőzött minden akadályt az eddigi folyamatok megértésénél, bizonyára nem esik nehezére az alprogram működésének önálló feldolgozása. Ha gondja támad, olvassa át még egyszer a 9. fejezetet.

Új fogalom a BASIC-ben:

Jelző (flag) használata

10.3 MEGOLDÁS LOGO-BAN

Először is tisztáznunk kell egy alapvető dolgot a LOGO adattárolásáról. A LOGO-ban a változónak numerikus, füzér vagy lista értékeket adhatunk. A listákat egymásba lehet ágyazni, vagyis a listák elemei újabb listák is lehetnek.
A listákat szögletes zárójelbe írjuk: [[a b] c] például olyan lista, amelynek első eleme egy két elemből álló lista [a b], második eleme pedig c. A listák elemeit üres helyekkel választjuk el egymástól.
A numerikus értékek olyan jelekből álló listák, amelyekkel számolni lehet. A füzérek olyan listák, amelyeknek nem kritériuma, hogy számolni lehessen velük. Numerikus értékeknél és szavaknál a szögletes zárójeleket elhagyjuk.

A listákkal kapcsolatos parancsok egy részével már megismerkedhettünk a „betűtölcsér"-nél és álcázott formában a „hétoldalú kocká"-nál is. Levághatjuk egy lista első elemét és vagy ezzel dolgozunk tovább, vagy a lista maradékával

(FIRST, BUTFIRST). Ugyanez érvényes az utolsó elemre is (LAST, BUTLAST). Ebben a fejezetben kellenek majd az inverz parancsok is, amelyekkel egy meglévő lista elejéhez vagy végéhez elemeket illeszthetünk (FPUT, LPUT). A 12. fejezetben a lista szemléletet még másképp is ki fogjuk használni. Itt most csak annyit, hogy rendező programunkat szinte változtatás nélkül alkalmazhatjuk füzérekből álló listákra is.

A betűtölcsér programban inputként listákat is adhatunk, csak a nyomtatási kép nem lesz szép. De ennél a programnál futási hibát eredményezne.

Hogyan rendezhetünk a LOGO-ban? Az alfabetikus sorrend vizsgálatához a LOGO-ban nem elég két jelet egyszerűen összehasonlítanunk a „nagyságuk" szerint, mint ahogy ezt a BASIC-ben vagy a PASCAL-ban tesszük. Az ASCII függvényt kell használnunk, amely az illető jel ASCII kódját adja meg. Így számokat kell rendeznünk növekvő sorrendben (ez megfelel az amerikai ABC sorrendjének).
Elöljáróban már említettük, hogy a LOGO-ban más algoritmust használunk, mint a másik két nyelvben. Végigmegyünk a beírt jelek listáján, megállapítjuk a legnagyobbat (ABC sorrendben a leghátul állót) és a lista végére helyezzük. Ezután a megmaradt jelek közül megint megkeressük a legnagyobbat és az utolsó előtti helyre tesszük. Ez a folyamat addig tart, amíg az összes jelet a helyes sorrendbe nem állítottuk. Itt is egy példán mutatjuk be a folyamatot. A rendezendő jelek listája a következő:

RENDEZES

A "legnagyobb" jel a Z betű. Ez kerül tehát az új lista végére

RENDEES Z

Megnézzük a RENDEES maradékszó betűit és újra a "legnagyobbat" választjuk ki. A keresett betű az S, tehát ez kerül az utolsó előtti helyre és a folyamatot a maradékszóval ismételjük meg. Ezt addig folytatjuk, amíg az összes jel elfogy, s ezzel a szót átrendeztük.
Az egyes lépések:

```
RENDEE  SZ
ENDEE  RSZ
EDEE  NRSZ
EDE  ENRSZ
D  EEENRSZ
  DEEENRSZ
```

A LOGO-ban most is, mint mindig a feladat megoldását eljárásokra bontjuk. Két ilyen eljárás alkotja az alapprogramot: a RENDEZETT és a LEGNAGYOBB eljárás.

```
TO RENDEZETT :H
 IF :H = "'' OP "''
 MAKE "H LEGNAGYOBB :H
 OP WORD RENDEZETT LEGNAGYOBB BL :H LAST :H
END

TO LEGNAGYOBB :H
 IF :H = "'' OP "''
 MAKE "X ASCII FIRST :H
 MAKE "Y ASCII LAST :H
 TEST :X > :Y
 IFF OP WORD FIRST :H LEGNAGYOBB BF :H
 IFT OP WORD LAST :H LEGNAGYOBB BL :H
END
```

Meghívhatja a rendezést a RENDEZETT szó beírásával, még mielőtt belekezdene az eljárások elemzésébe. Ekkor a RENDEZETT szó mögött egy üres helyet hagyva, a rendezendő karaktersorozatot meg kell adni. A LEGNAGYOBB eljárás a tervezett sorrendnek mindig csak egy lépését valósítja meg, ezért ezt az eljárást a RENDEZETT eljárásokból többször hívjuk. A LEGNAGYOBB eljárás minden híváskor úgy adja vissza a szót, hogy a legnagyobb betű a szó végére kerül.
A RENDEZETT eljárás során mindig levágjuk ezt a jelet és megőrizzük. A gép a megőrzést automatikusan elvégzi, ha az OUTPUT (OP) paranccsal találkozik. A szó maradékával (BL :M) ismét meghívjuk a LEGNAGYOBB eljárást. A leállási feltétel akkor teljesül, ha a RENDEZETT eljárás hívásakor üres szót találunk. Ekkor kezdődik a gépben megőrzött legnagyobb betűk kiírása. A WORD parancs úgy fűzi össze a betűket, hogy a ”legnagyobb” jel kerüljön a szó végére.

A RENDEZETT és a LEGNAGYOBB nevű eljárások összjátéka jobban megérthető, ha lépésről lépésre figyelemmel kísérjük az eljárások munkáját. Ezt a TRACE LOGO-parancs teszi lehetővé. Ha ezt a parancsot kiadjuk, akkor a gép minden parancsot kiír és csak azután hajtja végre, ha a RETURN billentyűt lenyomjuk. Így különösen jól látható, hogy a rendezési problémát visszavezettük az eggyel csökkentett hosszúságú szó rendezésére.
Normál üzemmódra a NOTRACE parancs beírásával térhetünk vissza.

Az alapprogramot szokás szerint kiegészítjük a felhasználónak szóló tájékoztatással. A RENDEZETT és LEGNAGYOBB eljárásokat változtatás nélkül vehetjük át.

```
TO RENDEZD
 RENDEZFEJ
 RENDEZFOLYAMAT
END

TO RENDEZFEJ
 ND
 TAB 5 [J E L E K R E N D E Z E S E] CR 1
 TAB 5 "= U 26 "= CR 3
 PRINT [KEREM, IRJON BE JELEKET] CR 2
 TAB 11 [A VEGEN NYOMJA LE A <RETURN>-T] CR 2
 S 2 MAKE "H INPUT
END
```

```
TO RENDEZFOLYAMAT
 CR 2
 MAKE "BALSZEL ( 40 - COUNT :H ) / 2
 TAB 15 [RENDEZETLEN] CR 1
 TAB :BALSZEL :H CR 2
 TAB 16 [RENDEZETT] CR 1
 TAB :BALSZEL RENDEZETT :H
 CR 1
END
```

Nem nehéz észrevenni, hogy a rendezési folyamatot a

RENDEZD

szó beírásával hívhatjuk meg. Ez az eljárás sorban hívja a RENDEZFEJ és RENDEZFOLYAMAT eljárásokat. Mint az előző példáinkban is, a RENDEZFEJ eljárás alakítja ki a képernyőt és kéri a bemenő adatokat a felhasználótól. A RENDEZFOLYAMAT révén a rendezetlen és rendezett jelsorozatot a képernyő közepére, szimmetrikusan írjuk ki (l. még 9. fejezet „betűtölcsér"). Ezen eljárások szerkezetét, illetve az itt használt LOGO-parancsokat illetően szinte semmi újjal nem találkoztunk.

Új fogalmak a LOGO-ban:

FIRST, LAST, FPUT, LPUT, ASCII, TRACE, NOTRACE

10.4 PASCAL PROGRAMOZÁS

A teljes program alig különbözik az alapprogramtól, akárcsak a BASIC megoldásnál. Ezért a Pascalban is eltekintünk az alapprogramtól.

A deklarációs részben először a **képernyőszélesség** nevű állandót definiáljuk. Értékét 80-nak választjuk. Ha másmilyen képernyőt használunk, akkor a 80 helyett a megfelelő számot kell beírni. Deklarálnunk kell azt a változót is, amelyikben a rendezendő füzért helyezzük el. Ehhez a 9. fejezetből ismert STRING típust használjuk, maximális hosszúságát a soronkénti jelek számával adjuk meg. Célszerű lenne a BASIC-hez hasonlóan egy tömbváltozót használni, hogy ebbe helyezhessük el a beírt füzér jeleit, és jó lenne egy indexmutató is, hogy ezek a jelek mindig elérhetők legyenek. A Pascalban az ARRAY adattípus szolgálja ezt a célt. További lehetőség a Pascalban, hogy definiálhatjuk a tömb elemeinek adattípusát is. Ezt kihasználva a következő módon definiáljuk a tömböt:

ARRAY [1..n] of char.

Ezzel egy egydimenziós tömböt határoztunk meg, amelynek az indexe 1-től n-ig terjed. Esetünkben **n** értékét a **kepernyoszelesseg** adja meg. A tömb elemei pedig csak karakteres változók lehetnek. Erre utal az *of char* jelölés. A típusdeklaráció is ugyanolyan szerkezetű, mint az állandók deklarációja, csak a TYPE szóval kezdődik. A Pascalban az ARRAY-ra vonatkozóan nincs olyan megkötés, mint a BASIC-ben a tömbökre; itt az index akárhonnan indulhat. Például így is definiálhatjuk:

ARRAY [5..9] of integer

Ezzel a tömb öt elemét érhetjük el az 5-ös és 9-es indexek között. Sőt, ha célszerű, betűindexeket is használhatunk:

ARRAY ['h'..'m'] of real

Az *of* utáni beírás határozza meg a tömb elemeinek adattípusát. Természetesen nem vagyunk szigorúan a négy már megismert típusra (integer, real, char, boolean) utalva, hanem bármilyen más, újonnan kitalált típust is definiálhatunk. Ezek a magyarázatok már érthetővé teszik a program definíciós részét:

```
program jelek_csoportositasa;
   { A program egy beirt 'szot' csoportosit, }
   { lexikografiai rend szerint.            }
const kepernyoszelesseg=40;
type  mezo              =array [1..kepernyoszelesseg] of char;
var   szo               :string [kepernyoszelesseg];
      jelmezo           :mezo;
      szohosszusag      :integer;
      i                 :integer;
procedure kepernyo;
   begin
       clrscr;
       writeln('JELEK CSOPORTOSITASA':(kepernyoszelesseg+31) div 2);
       writeln('====================':(kepernyoszelesseg+31) div 2);
       writeln;writeln;
       writeln('Kerem irja be a jeleket --');
       writeln('a RETURN lezarja a kiirast!':39);
       write('  ');readln(szo);
       writeln;writeln;
   end;
procedure mezo_bemutatas(x:integer; a:mezo);
   { x = szohosszusag }
var i:integer;
   begin
       for i:=1 to ((kepernyoszelesseg-x) div 2 ) do write(' ');
       { Beljebbtolas. }
       for i:=1 to x do write(a[i]);
       { Bemutatas.    }
       writeln
   end;
procedure csoportositas (var a:mezo);
var i                   : integer;
    csoportositott      : boolean;
   procedure felcserelni(var x,y:char);
   var z : char;
   begin
       z:=x ; x:=y ; y:=z
   end;
```

```
    begin { Csoportositas }
        repeat
           csoportositott:=true;
           for i:=1 to szohosszusag-1  do
              if a[i+1]<a[i] then
                 begin
                   felcserelni(a[i],a[i+1]);
                   csoportositott:=false
                 end;
        until csoportositott;
    end;
begin { FOPROGRAM .}
    kepernyo;
    szohosszusag:=length(szo);
    { A beirt szot kulonallo jelekre bontjuk, }
    { s a jelmezo [] mezobe atvisszuk.       }
    for i:=1 to szohosszusag do
        jelmezo[i]:=copy(szo,i,1);
        { A rendezetlen mezo bemutatasa. }
    writeln('rendezetlen': kepernyoszelesseg div 2+5);
    mezo_bemutatas (szohosszusag,jelmezo);
    { Jelmezo[]-t csoportositani. }
    csoportositas(jelmezo);
    { Csoportositott jeleket bemutatni. }
    writeln;writeln('csoportositott':kepernyoszelesseg div 2+4);
    mezo_bemutatas(szohosszusag,jelmezo)
end.
```

Haladjunk végig a főprogramon lépésről lépésre. Elsőként a KEPERNYO eljárást hívjuk meg. Ez az eddigi példáinkhoz hasonlóan épül fel. Itt írjuk be a rendezni kívánt füzért a **szo** nevű változóba. A következő lépésben ennek a jelsorozatnak a hosszát határozzuk meg.
Ezután a karakterláncot (füzért) egy FOR—TO—DO ciklussal karakterenként átvisszük a **jelmezo** nevű tömb elemeibe. Az erre vonatkozó függvényutasítás megfelel a BASIC-program hasonló részének. A füzért a gép, az ábra szerint, a beírás után kétszer írja ki. Először rendezés előtt, másodszor a rendezési folyamat után. Mivel mindkét esetben ugyanaz a folyamat, elég egy eljárás, a neve: MEZO_BEMUTATAS. A KEPERNYO eljárással ellentétben, a MEZO_BEMUTATAS eljárás két paramétert igényel: az egyik a rendezendő tömb, a másik a tömb elemeinek a száma, vagyis a tömb hossza.
Az átadandó paramétereket a Pascalban az eljárás híváskor, az eljárás neve után határozzuk meg, kerek zárójelek közé írva. Az eljárásfejben — szintén kerek zárójelek között — meg kell adni a változókat a típusukkal (köztük kettőspont) együtt, egymástól pontosvesszővel elválasztva, amelyek a paramétereket átveszik. Ennél a változóátadásnál értékeket adunk át, ezért e paramétereket *értékparaméterek*nek is nevezzük. (Szemben a *változóparaméterek*kel, amelyekről majd később beszélünk.) Az eljárás működtetéséhez szükségünk van még az i lokális változóra is, amelyet futóindexként használunk a FOR—TO—DO ciklusban. A MEZO_BEMUTATAS eljárás első utasítássora tolja a kiírást középre. A második utasítássorral íratjuk ki a jeleket. Az utolsó WRITELN utasítás pedig egy üres sort ír ki. Ezután a főprogramban a CSOPORTOSITAS eljárást hívjuk. Ez az eljárás veszi át a rendezendő tömböt (CSOPORTOSITAS(jelmezo))

A MEZO_BEMUTATAS eljárás paraméterdeklarációjától eltérően CSOPORTOSITAS eljárásban a deklaráció más fajtáját alkalmazzuk: az **a** változót a VAR szó előzi meg (VAR a: mezo). Ez azt eredményezi, hogy a

CSOPORTOSITAS (jelmezo)

eljáráshívás nem a **jelmezo** tartalmát adja át **a**-nak, hanem a **jelmezo** nevű változót az eljáráson belül **a**-nak nevezzük és meg is változtatjuk úgy, hogy a változtatás az egész programra érvényes legyen.
Ebben az eljárásban használunk még egy lokális változót (a FOR—TO—DO ciklus futóindexe) és egy logikai változót (boolean), amelynek értéke vagy csak igaz, vagy csak hamis lehet.
A CSOPORTOSITAS eljárás utasítás része pontosan a 10.1-ben leírt algoritmus szerint működik, amelyet már a BASIC-ben is használtunk. A külső ciklus (REPEAT—UNTIL) addig fut, amíg a **csoportosított** logikai változó nem kap igaz értéket. Mielőtt a belső ciklusba lépnénk (FOR—TO—DO), a logikai változó logikai értéke igaz (true) lesz. Ha a ciklusban cserére kerül sor, mert két szomszédos elem rossz sorrendben állt, akkor a logikai változó értéke ismét hamis (false) lesz, és így a rendezés folytatódik. Csak akkor marad az igaz (true), ha a belső ciklus lefutásakor már nem történt felcserélés. Ekkor kilépünk a REPEAT—UNTIL ciklusból. Két jel felcserélését a FELCSERELNI eljárás végzi el, amely lokális, vagyis CSOPORTOSITAS eljáráson belül definált eljárás. Ez az aleljárás felcseréli az **x**-ben és az **y**-ban átadott két karakter sorrendjét.
A felcserélendő karaktereket az eljárás hívásakor adjuk át az **x** és az **y** paramétereknek. Az eljárásban csak a két paraméter cserél helyet. (Ugyanúgy, mint azt a BASIC-részben leírtuk.) Hogy a felcserélés ne csak **x**-re és **y**-ra vonatkozzon, hanem a hívó CSOPORTOSITAS eljárás átadott értékeire is, mindkét paraméter elé a VAR szót írjuk. Így **x** és **y** *változóparaméter* lesz, vagyis csupán az átvett változók másik, „új” neve a FELCSERELNI eljárásban. Tehát a
FELCSERELNI (a[i],a[i+1])
eljáráshíváskor a[i] és a[i+1] tartalma cserélődik fel, nem pedig **x** és **y** tartalma.
A főprogram utolsó sorai ismét azt eredményezik, hogy a tömbmező nevű rendezett tömb a CSOPORTOSITOTT kiírás.

Új fogalmak a Pascalban:

értékek átadásának különféle lehetőségei eljárások között (értékparaméter, változóparaméter), saját változótípus létrehozása TYPE-pal, egész értékű osztás (DIV parancs).

11. HOLNAP UGYANÚGY

11.1 A FELADAT MEGHATÁROZÁSA

Már többször utaltunk arra, hogy ebben a fejezetben a programok betöltésével és tárolásával (mágneslemez, kazetta) foglalkozunk.

Hamar észrevehette, hogy a programok begépelésére fordított idő egyre inkább nő a gondolkodásra, megértésre és kipróbálásra fordított időhöz képest. Mivel a mai személyi számítógépek többsége a gép kikapcsolásakor elveszti a munkatár tartalmát, elsősorban a programok megőrzése okoz gondot.
Eleinte csak rögzítésről van szó. Később azonban a számítógép munkatárának tehermentesítése is szóba kerül, ugyanis esetleg úgy teletöltheti programokkal, hogy nem marad elég hely a tárban a program feldolgozásához.

A tárhelyet BYTE-ban mérjük. Például, a térfogatszámítási BASIC program (2.2 alfejezet) kb. 1000 byte-ot foglal le, tárigénye egészen kicsi. Az egészen kicsi számítógépekben is ennél jóval több a tárhely 16—32 Kb (Kb: kilobyte, 1 Kb $= 2^{10} =$ 1024 byte) áll rendelkezésre. A PC osztályú gépek pedig legalább 500 Kb-tal rendelkeznek.
A programokat mágneslemez helyett nyomtatóval kinyomtatva, papíron is tárolhatjuk. Az eddig bemutatott programok lehetnek példák az ilyen tárolásra.

Ennek a fejezetnek tehát az a feladata, hogy összefoglalja azokat az információkat, amelyek a számítógép típusától függetlenül, a tárolásra (lemez, kazetta), illetve a nyomtatásra vonatkoznak. Sajnos nagyok a különbségek a számítógéptípusok között, ezért ebben a témakörben sokszor kell a számítógép kézikönyvére hivatkozni.

11.2 BASIC PROGRAMOK BETÖLTÉSE, TÁROLÁSA ÉS KINYOMTATÁSA

Az alapvető utasítás, amellyel egy programot külső tárból beolvashatunk, a LOAD. Ahhoz, hogy a gép is tudja, hogy mit kell beolvasnia, meg kell adni a program nevét is. Itt már meg is szűnnek az azonosságok, majdnem minden gépnek másképp kell megadni a program nevét. Néhány lehetőség: a program

neve idézőjelben áll, vagy egy szóköz választja el stb. A szintaxist — így nevezik ezeket az írásmódbeli részleteket — a számítógép vagy a lemezmeghajtó kézikönyvéből ismerheti meg.
Lássunk két példát.

C 64-es: LOAD ”TERFOGAT”, 8

utasítja a számítógépet, hogy töltse be mágneslemezről a TÉRFOGAT nevű programot, feltéve, hogy a program ott rendelkezésre áll. Ha a 8-as helyett 1-est írunk, akkor a gép kazettáról kísérli meg a program beolvasását. A C 64-es egy különlegessége: ha egyszerre nyomjuk le a SHIFT és RUN/STOP gombokat, akkor a gép a kazettán megkeresi és betölti az első programot, valamint rögtön el is indítja.

APPLE: LOAD TERFOGAT D2

Utasítja a gépet, hogy a térfogat programot a második lemezmeghajtóról töltse be. Az első lemezmeghajtót 1-es számmal jelölik. A RUN TÉRFOGAT-tal betöltött program a betöltés után azonnal elindul. Ha azonban Ön a CP/M rendszer alatt működteti Apple gépét, akkor máris másképp hangzanak a parancsok. Remélhető, hogy a jövőben minden számítógép az általánosan használt operációs rendszerek (CP/M, MS—DOS) közül legalább az egyikkel rendelkezni fog, és használja majd a hozzá tartozó valamelyik MS—BASIC verziót (BASIC, BASICA, ...).

Itt a megfelelő parancs: LOAD ”TERFOGAT”

Ha már nem emlékezünk a program pontos nevére, először meg kell azt határoznunk. Ezt a lemez tartalomjegyzékének kiíratásával érhetjük el, amelyet sajnos, gyakorlatilag minden gépen másképpen találunk meg. A C 64-esnél LOAD ”$”,8-at kell beírnunk, majd a gép READY jelzése után LIST-et (ne felejtse el a RETURN gombot). Ha Ön MS—BASIC-et használ, a FILES parancs adja a lemez tartalomjegyzékét. Ha két lemezmeghajtója van, akkor CP/M vagy MS—DOS rendszerben esetleg meg kell adnia még a kívánt lemezmeghajtót is, tehát FILES”B:*.*”, ha a programlemez a B: lemezmeghajtóban van. Amint betölt egy programot a gép munkatárába, akkor a gép rendszerint törli az ott már bent lévő programot. A BASIC egyes változatainál lehetőség nyílik arra, hogy a rendelkezésére álló program mellé további programokat vagy programrészeket töltsünk. Az ezt végrehajtó parancsok neve — amennyiben van ilyen — MERGE, APPEND vagy hasonló. Azt tanácsoljuk, legyen óvatos ezek alkalmazásánál. Előzőleg készítsen minden érintett programról biztonsági másolatot.

A tárolást a betöltés mintájára végezheti. Az alapparancs a SAVE. Itt is kell a program neve, hogy később ezen a néven újra megtalálhassuk. Most, a betöltéssel ellentétben, olyan nevet választhat, amilyent csak akar. Ajánlatos viszont bizonyos óvatosság a különleges karakterek használatánál. Korlátozott a program nevének hossza is. CP/M-ben vagy MS—DOS-ban például csak 8 karakter hosszúságú lehet a név. Olvassa el tehát figyelmesen a számítógép kézikönyvének a névadással foglalkozó részét, különben meglepetések érhetik. Ha túl sok lesz a lemezen rögzített program, vannak olyan parancsok, amelyekkel a lemezen lévő állományokat törölni lehet. Az MS—BASIC-ben ilyen a KILL parancs, az APPLE-nél a DELETE és a C 64-esnél

OPEN 1,8,15,"S: FILE_NEV" parancs.

Végül szólnunk kell még a nyomtatásról. Itt is hiányzik az általános módszer és Ön a kézikönyvekre van utalva. A számítógép okozta meglepetéseken túl, számítania kell a nyomtató okozta meglepetésekre is. Például, nem feltétlen programhiba vagy egyéb zavar, ha a nyomtató a képernyőn megjelenő jelek közül néhányat másképp ad vissza.

Az MS—BASIC-nél az LLIST vagy az LPRINT paranccsal lehet átvinni a képernyőfeliratokat a nyomtatóra. Más BASIC változatok esetében a kézikönyvből kell tájékozódni. A C 64-es egyik lehetséges parancsa például:

```
OPEN 1,4:CMD1:LIST (RETURN)
```

Más adatok tárolásához meg kell ismerkednie az adatállomány fogalmával. Erre a 12. fejezetben talál majd példát.

11.3 LOGO PROGRAMOK BETÖLTÉSE, TÁROLÁSA ÉS KINYOMTATÁSA

A BASIC-ben a külső tárolók kezelésénél a számítógép és a külső tároló kézikönyve nyújtotta ismeretek állnak az előtérben. A LOGO ezzel szemben elrejti a gép sajátosságainak nagy részét. Ezáltal ugyanahhoz a művelethez minden LOGO ugyanazokat a parancsokat alkalmazza, ez természetesen leegyszerűsíti a külső tárolók használatát. A programokat mágneslemezről — a LOGO-nál a kazettás magnót nem ajánljuk — a READ parancs tölti be a tárba. Meg kell még adnia a betöltendő program nevét is. A LOGO az eléjük írt idézőjelről ismeri fel a neveket, így tehát a TERFOGAT nevű programot a

```
READ "TERFOGAT"
```

utasítással tölthetjük be a gépbe. A 12. fejezetben látni fogja, hogy a programnév betöltési parancsában változók értékét is megadhatja. A mágneslemezen tárolt állományok tartalomjegyzékét, amelybe a LOGO programok is beletartoznak, a CATALOG paranccsal találja meg. Ha CATALOG-ot ír be, és zárja a RETURN billentyűvel, a gép kiírja a tartalomjegyzéket. Megfigyelheti, hogy a program neve mögött minden LOGO programnál .LOGO vagy .LGO jelölés áll. Ezt a jelölést a gép tároláskor automatikusan kiteszi.

A TÉRFOGAT programunk tehát, TÉRFOGAT.LGO néven szerepel a tartalomjegyzékben. A betöltéskor azonban csak a "TÉRFOGAT" nevet szabad beírni, különben hibajelzést kapunk.

A program lemezre írására a SAVE parancsot használjuk. Itt is meg kell adni a kívánt programnevet, különben a

SAVE NEEDS MORE INPUTS

hibajelzést kapjuk.
A SAVE "TERFOGAT parancs a munkatár pillanatnyi tartalmát TERFOGAT.LGO nevű állományként helyezi el a lemezen.

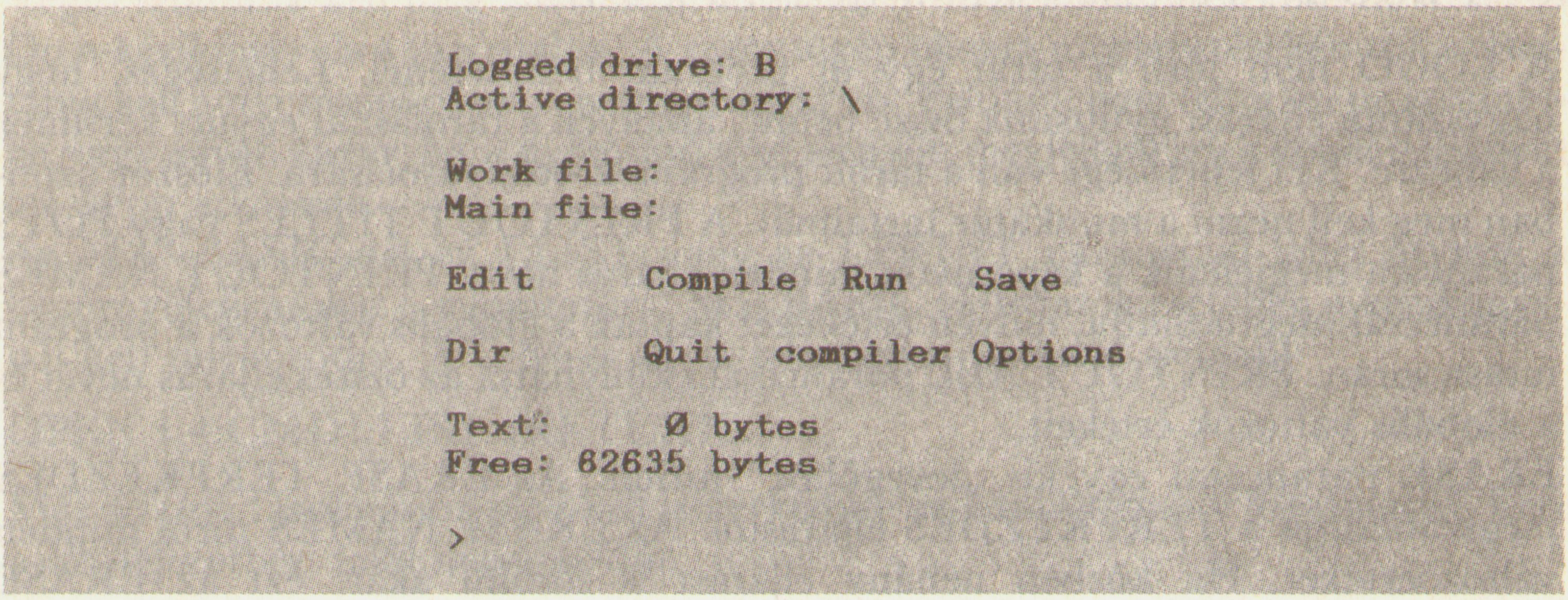
A LOGO nem törli a tárat egy új program betöltésénél. Lehetséges, hogy nagyszámú eljárást gyűjtsünk össze benne, amelyek a tárolásnál együtt kerülnek a lemezre. Két lehetősége van a tárolt programok befolyásolására. Először azonban meg kell nézni a munkatár tartalmát. A PRINTOUT TITLES vagy POTS parancsra megjelenik a képernyőn minden eljárás neve. PRINTOUT NAMES paranccsal megtudhatja, milyen értékeket kaptak a globális változók a program futása során. PRINTOUT ALL paranccsal előhívhatja az összes eljárás nevét és a globális változók értékeit.
ERASE paranccsal egy kis „levegőhöz" juthat. Az ERASE TERFKEZDES megszabadítja a TERFKEZDES eljárástól. Az ERASE NAMES töröl minden változóértéket. Ha közben néhány eljárást is törölni akar, ezt az ERASE [VTERFKEZDES SZAMITAS]-sal érheti el. Ilyenkor a gép törli a szögletes zárójelbe zárt eljárásokat. Az ERASE ALL parancsra egyszerre minden programtól megszabadul. Ugyanezt a hatást éri el a GOODBYE beírásával is.
Ha a most megmagyarázott parancsokkal rendet rak a tárban, csak a kívánt eljárások kerülnek tároláskor a lemezre. A másik mód, hogy csak bizonyos eljárásokat vigyünk ki lemezre a SAVE "TERFOGAT [V TERFKEZDES SZAMITAS] parancs. Ezzel a tár tartalmától függetlenül csak a zárójelben álló eljárásokat írja ki a számítógép TERFOGAT.LOGO néven a lemezre.

Ha meg akar szabadulni valamelyik, a lemezen levő LOGO állománytól, azt az ERASEFILE paranccsal érheti el. ERASEFILE "TERFOGAT" törli a lemezen az imént tárolt állományt.

A LOGO-ban a nyomtató használata különösen egyszerű, ha a számítógépe be van állítva. PRINTER parancs kiadása után minden, a képernyőn megjelenő felirat a nyomtatón is kinyomtatódik. A POTS hatására most papíron is megjelenik a rendelkezésére álló eljárások listája. Ha elindít egy programot, a gép a program menetét is kinyomtatja. Számítania kell ekkor is meglepetésekre, különösen akkor, ha a nyomtató és a számítógép illeszkedése nem tökéletes. A NOPRINTER parancs megszünteti a nyomtatást.

11.4 PASCAL PROGRAMOK BETÖLTÉSE, TÁROLÁSA ÉS KINYOMTATÁSA

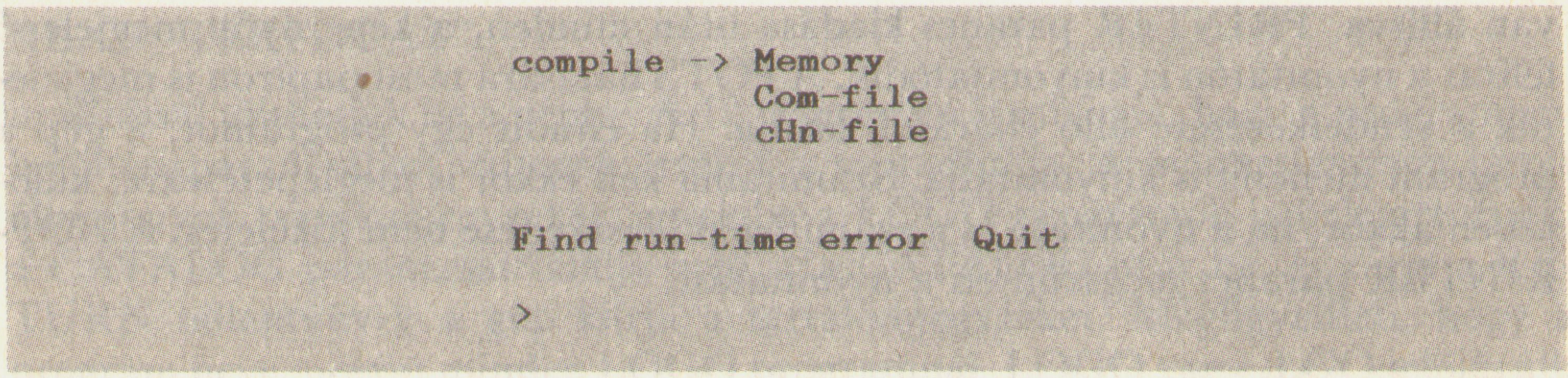
Ha a TURBO—Pascallal dolgozik, a betöltést és a tárolást az indításkor megjelenő menü irányítja:



```
Logged drive: B
Active directory: \

Work file:
Main file:

Edit     Compile  Run   Save

Dir      Quit  compiler Options

Text:     0 bytes
Free: 62635 bytes

>
```

Nem akarjuk a kézikönyvet helyettesíteni, ezért csak röviden szólunk erről a menüről. Ha E-t (Edit) gépel be, az editorba jut és ott megváltoztatja vagy megíratja programját. Control KD-vel ismét elhagyhatja az editort. A C-vel (Compile)az editorban létrehozott programot lefordítja, kompilálja. R-rel (Run) elindítja a programot, S-sel (SAVE) mágneslemezen tárolja azon a néven, amelyet a Work File-nál adott meg. Tartalomjegyzéket D-vel (Dir) kap és Q-val (Quit) kilép a TURBO—Pascalból.

Ha olyan programot akar készíteni, amely a TURBO—Pascal nélkül is működőképes, akkor a kompilált programját mágneslemezen kell elhelyeznie. Válassza ehhez az O betűt (compiler Options), így egy újabb menüt kap.



```
compile -> Memory
           Com-file
           cHn-file


Find run-time error  Quit

>
```

Itt a fordítás három lehetősége közül választhat. Minket csak az első kettő érdekel. Az M-mel (Memory) meghatározza, hogy a fordítás a gép munkatárába történjen, ill. C-vel (Com-file) mágneslemezre íratja — azonos néven, de .COM kiterjesztéssel —, ahonnan a név megadásával elindíthatja a programot anélkül, hogy a TURBO-rendszert betöltötte volna.

Amilyen egyszerű a TURBO—Pascal kiszolgálása, olyan bonyolult lehet esetleg más fordítóké. Különösen a „nagy értékű”, drága Pascal fordítók adnak rengeteg munkát a felhasználónak. Cserébe viszont komplex kiszolgálást nyújtanak. Tanulmányozza át minden esetben a kézikönyvét és a fordítójának a szolgáltatásait.

A forrásprogram kinyomtatásához általában azt a szövegszerkesztőt használják, amellyel a forrásprogramot megírták. A TURBO—Pascal önmagában nem rendelkezik nyomtatási lehetőséggel, de van viszont egy TLIST nevű programja, amellyel kinyomtathatja a forrásprogramot.

12. PILLANTÁS A JEGYZETFÜZETBE

12.1 A FELADAT MEGHATÁROZÁSA

A tapasztalatok szerint az állományok (file-ok) használata mindig nehézséget okoz. Igyekeztünk az állománykezeléssel kapcsolatos problémákat egy feladatba összehozni. Példaként egy erősen leegyszerűsített címjegyzékállományt választottunk. Ugyanígy választhattunk volna ehelyett egy szövegekből, vagy mérési adatokból álló állományt is. Így tehát nem a feladat, ill. annak lehetőleg elegáns megoldása áll az előtérben, hanem sokkal inkább az állományok külső adattárolókon, esetünkben mágneslemezen történő kezelésének a bemutatása. A feladat megválasztásánál elkerülhetetlen volt az önkényesség.

Mindenekelőtt tisztázzunk röviden néhány fogalmat. Egy állományt úgy kell elképzelnünk, mint egy kartotékokban tárolt információhalmazt, mint például egy cédulákkal teli dobozt. Általában az összes adatkártya szerkezete azonos. Az informatika szóhasználata szerint az *állomány:* az a doboz, amelyben a kártyák vannak. Az egyes cédulák vagy kártyák az állomány *adattételei* (rekordjai). A kártyákon a bejegyzéseknek fenntartott helyek, úgy mint név, cím, vevő száma stb. az adattételek *mezői*. Mindezeket az összegyűjtött információkat valamilyen értelmes módon kellene kezelnünk.

Ha egy új vevőt akarunk felvenni az adattárba, akkor ez a kartotékdobozos rendszer esetén minden nehézség nélkül megoldható: írunk egy új kartotéklapot a vevő adataival, és eltesszük a doboz megfelelő helyére. Egy mágneslemezen lévő adatállománnyal ez már nem ilyen egyszerű.

Példánk egy egyszerű adatállomány rendezését és kezelését mutatja be a születésnapok listáján.

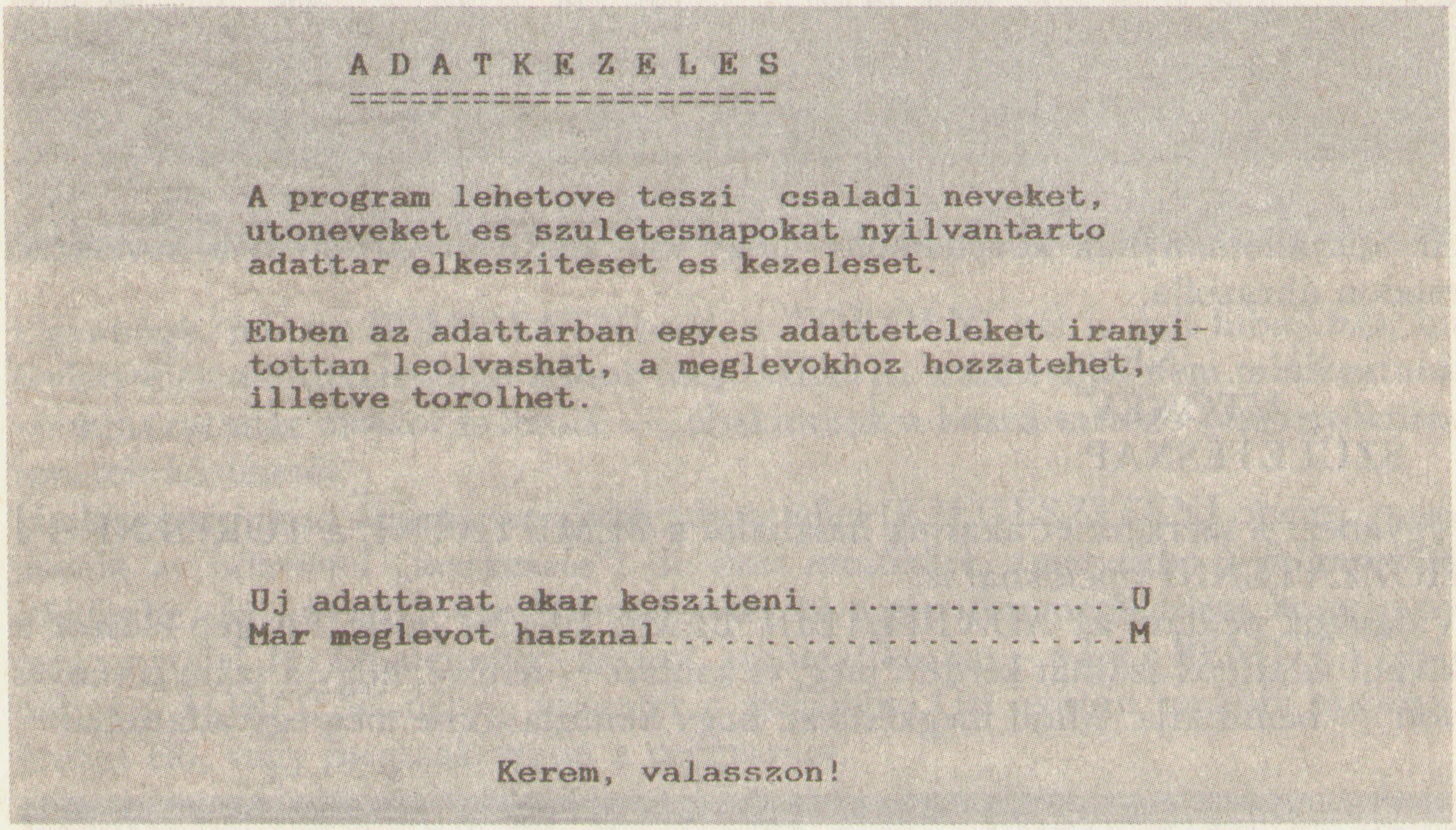
Az adatállomány egy adattétele három mezőt tartalmaz: családi név, utónév, születési idő. Szem előtt tartva az egyszerűsítést, csak három funkciót kívánunk meg:

— az adattételek bemutatása,
— az adattételek törlése,
— az adattételek kiegészítése.

Nem foglalkozunk olyan fontos dolgokkal, mint: egy bizonyos adattétel kikeresése, alfabetikus rendezés stb. De mint már mondtuk, nem a feladat teljes megoldása a lényeges, hanem annak a bemutatása, hogy hogyan dolgozhatunk mágneslemezen lévő adatokkal.

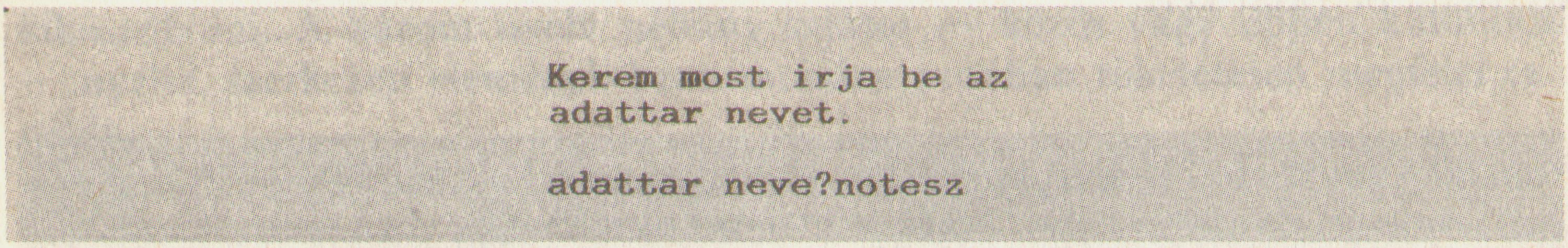
Léteznek olyan nagyteljesítményű adatállomány-kezelő programok, amelyek leírásához egy egész könyvre lenne szükség. Ezért ilyen programokkal itt nem foglalkozhatunk.

Mint az összes eddigi példánknál, a feladat leírásánál a hozzá tartozó képernyőábrára támaszkodunk. Ez már egy erősen leegyszerűsített adatállomány esetén is igen terjedelmes:



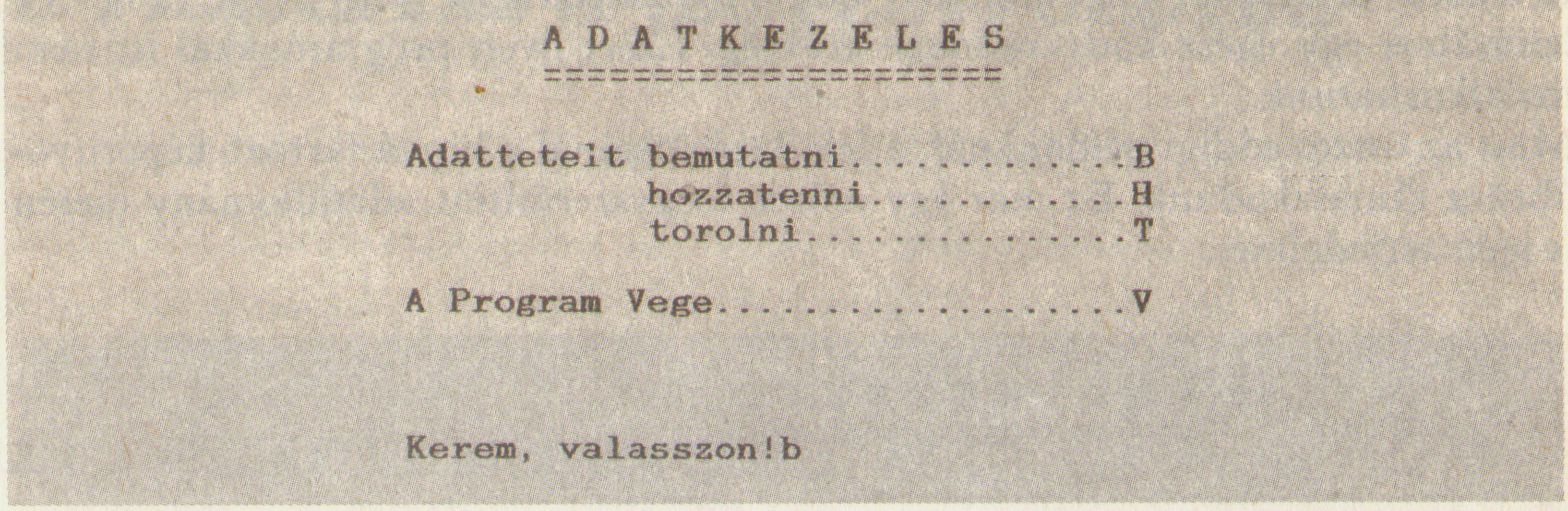
```
              A D A T K E Z E L E S
              =====================


       A program lehetove teszi  csaladi neveket,
       utoneveket es szuletesnapokat nyilvantarto
       adattar elkesziteset es kezeleset.

       Ebben az adattarban egyes adatteteleket iranyi-
       tottan leolvashat, a meglevokhoz hozzatehet,
       illetve torolhet.




       Uj adattarat akar kesziteni...............U
       Mar meglevot hasznal......................M



                   Kerem, valasszon!
```

Az első képernyő röviden ismerteti a program működését. Ezután a gép megkérdezi Öntől, hogy új adatállományt akar-e elkészíteni, vagy a már meglévőt akarja használni. Miután választott, megjelenik a második képernyő, amelyen a gép kéri az adatállomány nevét. A mágneslemezen ezen a néven lehet (majd) az adatállományt megtalálni.



```
                    Kerem most irja be az
                    adattar nevet.

                    adattar neve?notesz
```

Ha az állomány nevét beírtuk, a harmadik képernyőn egy menü jelenik meg, amelyből kiválaszthatja, hogy mit kíván tenni a továbbiakban: befejezi-e a munkát, vagy tovább dolgozik.

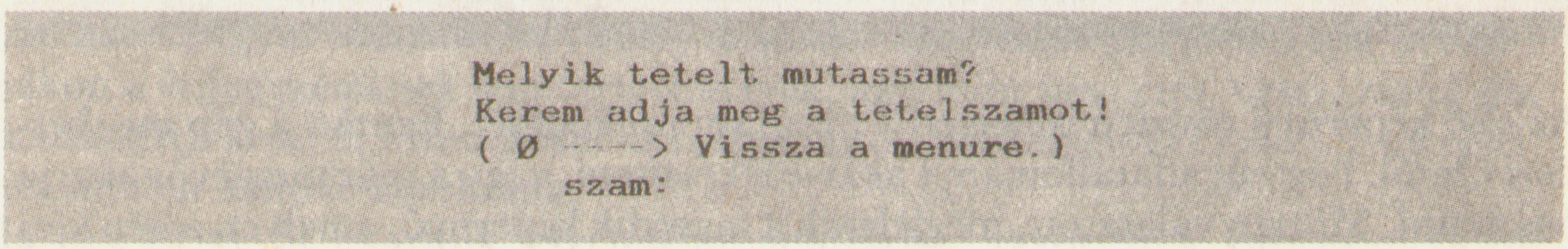


```
                    A D A T K E Z E L E S
                    =====================

          Adattetelt bemutatni...........B
                     hozzatenni..........H
                     torolni.............T

          A Program Vege.................V


          Kerem, valasszon!b
```

Az adatállománynak követelmény, hogy egy adattétel tartalmát a következő módon ábrázolja:

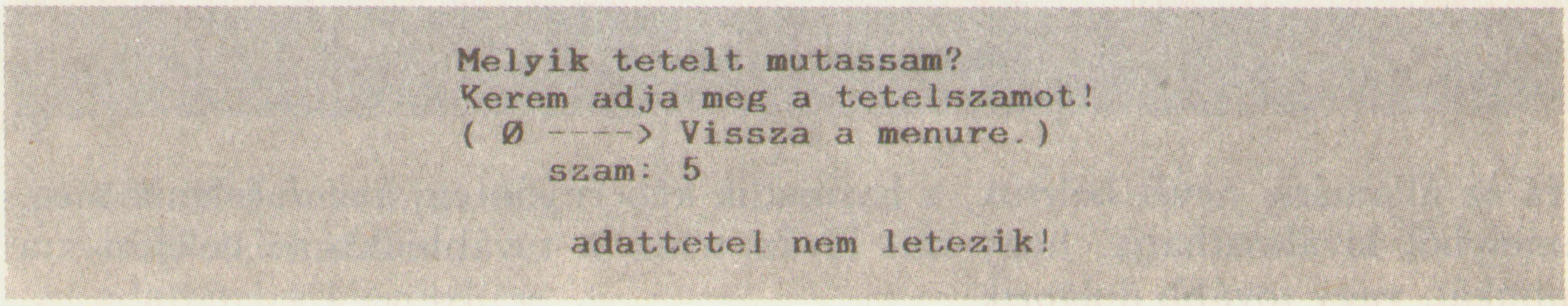
NÉV ...
UTÓNÉV ...
SZÜLETÉSNAP ...

Ugyanezt a szerkezetet akarjuk használni a BEMUTATNI, a TOROLNI és a HOZZATENNI opcióknál is.
Válasszuk például az ADATTETELT BEMUTATNI opciót. A gép először a kívánt adattétel számát kérdezi meg, és azután — feltéve, hogy a szám érvényes volt — bemutatja. Majd megkérdezi, hogy bemutasson-e még egy adattételt.



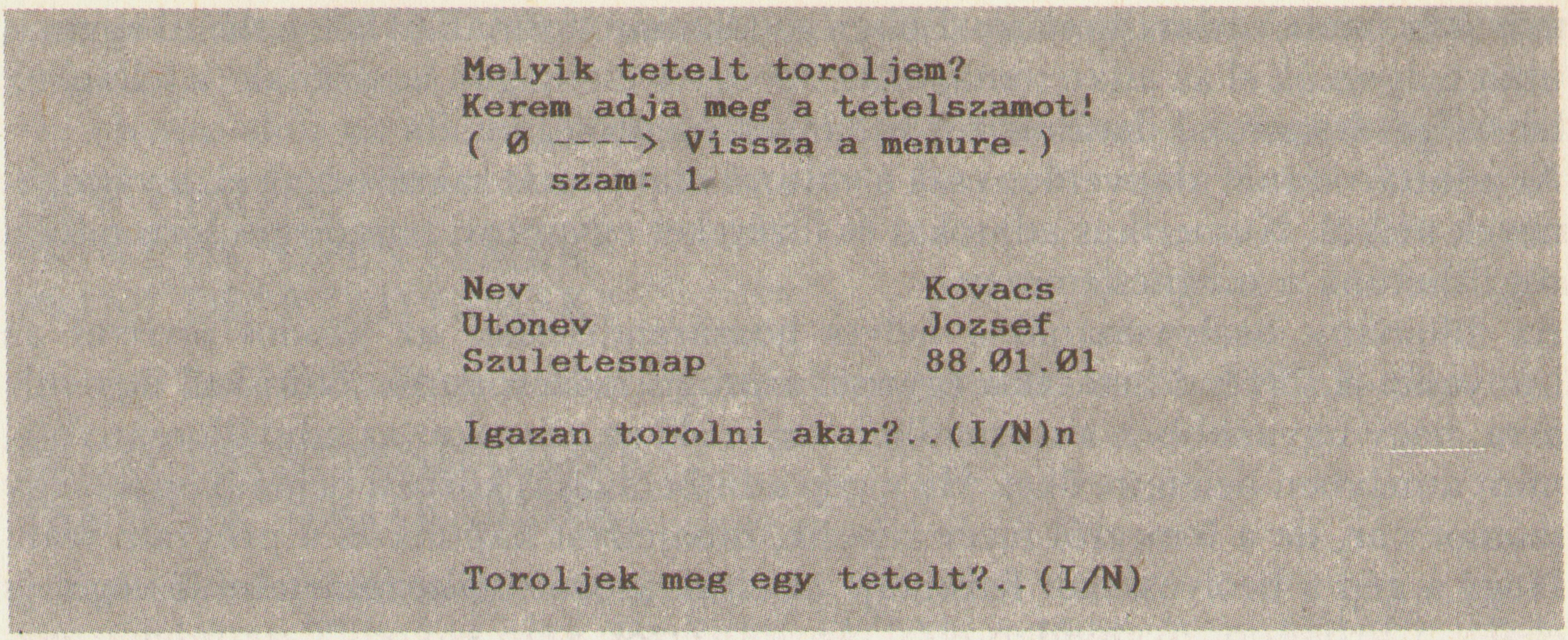
```
Melyik tetelt mutassam?
Kerem adja meg a tetelszamot!
( 0 ----> Vissza a menure.)
     szam:
```

Ha a beírt számon nincs adat, erről a gép tájékoztat a képernyőn és másik adattétel beírását kéri.



```
Melyik tetelt mutassam?
Kerem adja meg a tetelszamot!
( 0 ----> Vissza a menure.)
     szam: 5

      adattetel nem letezik!
```

Ugyanilyen a képernyőfelépítés a TOROLNI opciónál is. Itt is megkérdezi a gép a törlendő adattétel számát. Ha ezt a számot beírtuk, bemutatja az adattételt, majd megerősítést kér, hogy ténylegesen törölni akarunk-e. Ha a felhasználó válasza I, a gép törli az adattételt és ezt jelenti is a felhasználónak. Ezután megkérdezi, hogy töröljön-e még egy adattételt.



```
          Melyik tetelt toroljem?
          Kerem adja meg a tetelszamot!
          ( 0 ----> Vissza a menure.)
              szam: 1


          Nev                     Kovacs
          Utonev                  Jozsef
          Szuletesnap             88.01.01

          Igazan torolni akar?..(I/N)n



          Toroljek meg egy tetelt?..(I/N)
```

Ugyanúgy, mint a BEMUTATNI-nál, a TOROLNI-nél is van hibajelzés, ha érvénytelen adattételszámot adunk meg. Mivel itt semmi újjal nem találkoztunk — mint ezt már máskor is tettük —, eltekintünk a hozzá tartozó képernyőkiírás ismételt közlésétől.
Ha kiegészítjük az állományt újabb adattétellel (KIEGÉSZÍTENI opció), megjelenik az adattétel „szerkezete”, de üres mezőkkel, amelyekbe a felhasználó írhatja be a kívánt adatokat. Ha ezzel elkészül, a gép itt is megkérdezi, hogy lesz-e még egy tétel. Ha Ön nemmel válaszol, automatikusan — mint a BEMUTATNI-nál és a TOROLNI-nél — visszatér a menühöz. Ott azután további opciókat hívhat elő, vagy megszakíthatja a programot.

Végül még egyszer szeretnénk felhívni a figyelmét, hogy a feladat megoldása távolról sem teljes. Bizonyos opciókat, pl. a megváltoztatást, össze lehet állítani a rendelkezésre álló opciókból. Másokat, mint pl. a keresést és a rendezést tudatosan hagytuk ki. A megoldás ezen hiányosságaihoz még továbbiak is csatlakoznak, amelyekre egy kezdő programozó nem is gondolhat. Előfordulhat, főleg hibás beírások esetén, hogy a program nemkívánt módon reagál, esetleg szét is eshet. Az ilyen jellegű gondokat csak sokkal terjedelmesebb programmal lehetne kiküszöbölni. A három nyelv néhány sajátos — közös vagy éppen különböző — nyelvi, szerkezeti elemének bemutatására azonban tökéletesen megfelel ez a példa.
Alapprogram írásának egyik nyelvben sincs értelme, hiszen felhasználói tájékoztatások nélkül még a program jó ismerője is kísérletezésre és tévedésekre lenne ítélve. A program néhány elemét mindenesetre sokkal részletesebben ismertetjük, mint az előző fejezetekben.

12.2 KIDOLGOZÁS BASIC-BEN

Mint már mondtuk, ennél a feladatnál nincs értelme alapprogramot írni. Ehelyett jóval fontosabb a BASIC mágneslemezes tárolási szerkezetének az ismerete. Az adattárolásnak az a formája, amelyet minden BASIC-változat ismer, a szek-

venciális (soros) adatállomány. Ebben az állományformában közvetlenül egymás után helyezzük el az adatcsomagokat. Az egyes csomagok hosszának (a jelláncok lehetséges hosszának határai között) nincs jelentősége.
Az adattárolásnál alapvetően csak a változók tartalmát raktározzuk el, a változónevek nélkül. Numerikus adatokat és füzéreket egyaránt, tetszőleges sorrendben rögzíthetünk a mágneslemezen.
Az állomány beolvasásánál a helyes hozzárendeléshez az azonos sorrend az irányadó. A sorrend betartása egyedül a programozón múlik. Neki kell ügyelnie arra, hogy ugyanebben a sorrendben vegye elő az adatokat az adatállományból, mint amelyben beletette őket. Ha a gép adatbeolvasás közben például éppen egy számot vár, de a lemezről füzér érkezik, hibajelzést kapunk és a program leáll. Hogy a gép a beolvasáskor felismerje az elraktározott változótartalmak végét és elejét, a programozó minden elraktározott érték után egy elválasztó jelet tesz. A gép csak erről az elválasztó jelről ismeri fel, hogy hol kezdődik új változóérték.

Elvileg két módszer létezik egy adatállomány adatainak feldolgozására. Az egyik lehetőség, hogy az egész adatállományt a gép tárjába töltjük, ott feldolgozzuk, azután újra eltároljuk. A másik lehetőség, hogy mindig egy-egy adattételt olvasunk be, feldolgozzuk és újra tároljuk. Az állománykezelés bemutatására a BASIC-ben e második lehetőséget választottuk. Sok más módszer is van, amellyel az állománykezelést megvalósíthatjuk.

Ahhoz, hogy egy állományból olvashassunk vagy egy állományba írhassunk, kapcsolatot kell létesítenünk a számítógép és munkatársa, a lemezmeghajtó között — szaknyelven: meg kell nyitnunk egy adatátviteli csatornát.
Ahhoz, hogy egyidejűleg több csatornát nyithassunk meg, a csatornákat meg kell számozni. Csak így léphetünk kapcsolatba egy meghatározott csatornával. Ezt a kapcsolatot az OPEN paranccsal létesítjük. A parancs után kell a kapcsolatlétesítés tárgyát, célját és útját megjelölni.
Ha mágneslemezre akarunk adatokat vinni, az IBM PC BASIC-nél az

```
OPEN "CIMEK" FOR APPEND AS #1
```

parancsot használjuk. Ha a CÍMEK nevű állomány még nincs a mágneslemezen, akkor ezzel a paranccsal megnyitjuk, létrehozzuk. Amint ezt az utasításból kiolvashatjuk, a CÍMEK nevű állományt nyitottunk meg, amelyhez adatokat kívánunk hozzáfűzni (APPEND), az 1-es jelű adatátviteli csatornán keresztül.
Más BASIC változatokban ez a parancs másképp szól. A C 64-esen a következő parancsot kell használni:

```
OPEN 1,8,6,"CIMEK,S,A"
```

amelyben az S jelzi, hogy szekvenciális állományról van szó, az A pedig az APPEND rövidítése.
Ha az állományt olvasni akarjuk, akkor az A helyett R-t (read), az IBM BASIC-eknél pedig az APPEND helyett INPUT-ot kell írni.
Ha Apple gépen akar szekvenciális állományokkal dolgozni, akkor célszerű bevezetnie egy változót, például D$-t, CHR$(4) értékkel, mert minden állomány

parancs előtt a PRINT D$; utasítással jeleznie kell, hogy mágneslemezzel akar dolgozni.
Ha olvasni akar az állományból, akkor ehhez a következő parancsok kellenek:

```
D$ = CHR$(4)
PRINT D$;”OPEN CIMEK”
PRINT D$;”READ CIMEK”
```

Ha már kész állományhoz akar adatokat hozzátenni, azt az APPLE-nél a

```
PRINT D$;”APPEND CIMEK”
PRINT D$;”WRITE CIMEK”
```

parancsokkal teheti meg.
Ha viszont az állomány még nem is létezik, akkor először össze kell állítania:

```
PRINT D$;”OPEN CIMEK”
PRINT D$;”WRITE CIMEK”
PRINT D$;”CLOSE CIMEK”
```

megnyitott állományhoz a munkatárból az ismert PRINT paranccsal tudunk adatokat hozzátenni. Kiegészítésképpen azonban még meg kell adnunk, hogy az adatokat nem a képernyőre akarjuk kiírni, hanem az előzőleg megnyitott adatátviteli csatornán keresztül a lemezre. A parancs

```
PRINT#1,PELDA
```

a PÉLDA változó tartalmát az 1-es csatornán keresztül a mágneslemezre viszi. A mágneslemezről ennek megfelelően az INPUT paranccsal olvasunk.

```
INPUT#1,PELDA
```

paranccsal a következő adatcsomagot olvassuk be a lemezről, az 1-es csatornán át a PÉLDA változóba. Mint már említettük, ilyenkor természetesen számnak kell állnia a lemezen és nem karakterfüzérnek. Ha minden műveletet elvégeztünk, CLOSE-zal zárjuk le a csatornát.
Az APPLE-nél viszont, ha a szokásos PRINT paranccsal nyitottuk meg az állományokkal, minden adat *csak* a lemezre kerül, és az INPUT parancs minden adatot *csak* a lemezről hoz be. Amennyiben az adatforgalom befejeződik, a PRINT D$; ”CLOSE CIMEK”-kel le kell zárni a csatornát és csak azután dolgozhat ismét rendesen a PRINT és INPUT parancsokkal. A két parancs ilyen használata azt eredményezi, hogy minden programot át kell alakítani, amelyben egyidejűleg akarunk képernyőre és lemezre íratni, vagy onnan beolvastatni adatokat. Ezeket a módosításokat a program kialakításánál nem vettük figyelembe.
Az összes itt leírt parancs alkalmazásával találkozhat a megfelelő programrészekben.

A program szerkezete az eddigi példákhoz képest természetesen mutat néhány változást:

10—99-es sorig	programnév és a programot leíró kommentár (úgy, mint eddig)
100—299-es sorig	előzetes tájékoztatás a felhasználónak és választási lehetőségek (úgy, mint eddig)
300—999-es sorig	főrész a választási lehetőségekkel: bemutatás, kiegészítés, törlés, befejezés
1000—2340-es sorig	alprogramok
1000-estől	ismertetés, útmutató
1300-astól	állománynév meghatározása, új állomány létrehozása
1500-astól	menü
1700-astól	olvasó program
2100-astól	törlési program

Az adattételeket be-, ill. hozzáírni a főprogramban lehet. Ezzel az áttekintéssel könnyen tájékozódhat a programban. Célszerű, ha először a részfunkciók kapcsolódására összpontosít. Ehhez nézzük meg a program első részét a 299-es sorig.

```
10 REM Adatkezeles
20 REM ---------------------------------------
30 REM Egy csaladai es utonevekbol allo szuletesi
40 REM listat akarunk vezetni.A program tegye le-
50 REM hetove adattetelek iranyitott bemutatasat,
60 REM torleset es kiegesziteset.
70 REM ---------------------------------------
99 :
100 REM beirasi kepernyok es az egyes beirasok
101 REM ====================================
110 GOSUB 1000     :REM Magyarazatok
120 INPUT BEIRAS$  :REM uj vagy meglevo allomany
130 GOSUB 1300     :REM allomanyok nevei
140 REM Menu bemutatasa es valasztas, vegrehajtas
150 GOSUB 1500     :REM menu-kepernyo
160 INPUT VALASZTAS$  :REM felhasznaloi valasztas
170 REM ugras a valasztott programreszhez
180 IF (VALASZTAS$="b") OR (VALASZTAS$="B") THEN 340
190 IF (VALASZTAS$="h") OR (VALASZTAS$="H") THEN 500
200 IF (VALASZTAS$="t") OR (VALASZTAS$="T") THEN 650
210 IF (VALASZTAS$="v") OR (VALASZTAS$="V") THEN 810
220 REM ELSE rossz beiras
230 GOTO 140       :REM vissza a Menure
299 :
300 REM FORESZ allomany lekerdezese
301 REM ====================
310 REM Ez a resz negy reszre oszlik, a negy
320 REM menupontnak megfeleloen:
330 REM BEMUTATAS, HOZZATETEL, TORLES, VEGE.
```

A 110-es sorban meghívott alprogram (1000—1160) előállítja az 1. képernyőt, a 120-as sor fogadja a felhasználó válaszát, hogy új állományt akar-e készíttetni, vagy a meglévőt bővíteni.

A felhasználó válasza után a 130-as sorban a 2. képernyőt hívjuk. Az alprogram (GOSUB 1300) megkérdezi az állománynevet, kiértékeli a 120-as sorban beírt választ, és ha kell, új állományt készít.

A 150-es sor azt az alprogramot hívja, amely a menüt készíti el. Az ott kért választást a 160-as sor fogadja, majd a 180—210-es sorokban kiértékeli és a választás szerinti programrészre ugrik. Ha rossz billentyűt nyomott le a felhasználó, akkor a négy lehetőség egyike sem valósul meg (180—210 sor) és a program a 140-es sorra lép vissza, ismét hívja a menüt.
Az alprogramokat két részletben közöljük:

```
996 :
997 :
998 REM ************* ALPROGRAMOK *************
999 :
1000 REM ***** AP: Magyarazatok *****
1010 CLS
1020 PRINT TAB(6)"A D A T K E Z E L E S"
1030 PRINT TAB(6)"====================="
1040 PRINT:PRINT
1050 PRINT"A program lehetove teszi egy csaladi neveket, "
1060 PRINT"utoneveket es szuletesnapokat nyilvantarto    "
1070 PRINT"allomany elkesziteset es kezeleset.":PRINT
1080 PRINT"Ebben az allomanyban az egyes adatteteleket beol-"
1090 PRINT"vashatja, torolheti, avagy a meglevokhoz      "
1100 PRINT"hozzatehet."
1110 FOR I=1 TO 5:PRINT:NEXT I
1120 PRINT"   Uj allomany letrehozasa................U"
1130 PRINT"   Meglevo allomany hasznalata............M"
1140 FOR I=1 TO 3:PRINT:NEXT I
1150 PRINT TAB(14);"Kerem valasszon ";
1160 RETURN


1299 :
1300 REM ***** AP: Allomany megnyitasa *****
1310 REM az allomany nevenek bekerese
1320 CLS:FOR I=1 TO 5 :PRINT :NEXT I
1330 PRINT"Kerem most adja meg az allomany"
1340 PRINT"nevet.":PRINT
1350 PRINT"Allomanynev: ";:INPUT DNAME$
1360 REM ha mar megvan az allomany, semmit sem kell tenni
1370 IF (BEIRAS$="b") OR (BEIRAS$="B") THEN 1410
1380 REM ELSE uj allomany letrehozasa
1390 OPEN DNAME$ FOR APPEND AS #1
1400 CLOSE 1
1410 RETURN


1499 :
1500 REM ***** AP: Menu bemutatasa *****
1510 CLS
1520 PRINT TAB(6);"A D A T K E Z E L E S"
1530 PRINT TAB(6);"====================="
1540 FOR I=1 TO 7 :PRINT:NEXT I
1550 PRINT"Adattetelt bemutatni ..................... B"
1560 PRINT"           hozzatenni ..................... H"
1570 PRINT"           torolni ........................ T"
1580 PRINT
1590 PRINT"A program VEGE ............................ V"
1600 PRINT:PRINT:PRINT
1610 PRINT"           Kerem valasszon!";
1620 RETURN
```

Az 1000-es, az 1300-as és az 1500-as sorokban kezdődő alprogramok nem sok újat tartalmaznak, ezért foglalkozunk inkább a főrésszel: a négy lehetséges választással és megvalósításaikkal. Innentől kezdve a választástól függően más-más képernyőt kapunk.

```
340 REM BEMUTATAS ----------------------------
350 CLS:PRINT
360 PRINT"Melyik adattetelt mutassam be?"
370 GOSUB 1700      :REM Tetelszam be, adat ki
380 IF SZAM=O THEN 140  :REM vissza a menure
390 REM ELSE
400 IF HIBA=1 THEN 360  :REM nem ervenyes tetelszam
410 REM ELSE
420 PRINT"Bemutassak meg egy adattetelt?  (I/N)";:INPUT BEIRAS$
430 IF (BEIRAS$="i") OR (BEIRAS$="I") THEN 350
440 REM ELSE vissza a menure
450 GOTO 140
```

1. választás: BEMUTATAS

Ez a programrész a 340-es sorban kezdődik és a 370-es sorban TETEL BEMUTATASA alprogramot hívja meg.

```
1699 :
1700 REM ***** AP: Tetel bemutatasa *****
1710 REM Tetelszam be, adat ki
1720 PRINT"Kerem adja meg a tetelszamot!"
1730 PRINT TAB(10);"( 0 ---> Vissza a menure! )"
1740 PRINT"    szam: ";:INPUT SZAM
1750 IF SZAM=O THEN 2000 : REM vissza a menure
1760 REM ELSE a kivant adattetelt megkeresni es beolvasni
1770 OPEN DNAME$ FOR INPUT AS #1
1780 HIBA=0
1790 REM addig keresni, amig a kivant tetelt meg nem talaljuk
1800 FOR I=1 TO SZAM
1810   IF EOF(1)=0 THEN 1850 : REM meg nem ertuk el az allomany veget
1820     REM ELSE allomany vege ==== hibajelzes
1830     HIBA=1
1840     I=SZAM : GOTO 1860   :REM keresest abbahagyni
1850   INPUT #1,CSNEV$,UNEV$,SZNAP$ :REM tetel be
1860 NEXT I
1870 REM  hiba=0 -----> megvan a tetel
1880 REM  hiba=1 -----> nincs meg
1890 CLOSE 1
1900 IF HIBA=0 THEN 1940 :REM ervenyes tetelszam, adat be
1910 REM ELSE elertuk az allomany veget, hibajelzes
1920 PRINT:PRINT"  Adattetel nem letezik!":PRINT
1930 GOTO 2000
1940 REM adattetel megvan, bemutatni!
1950 PRINT:PRINT
1960 PRINT"Nev                ";CSNEV$
1970 PRINT"Utonev             ";UNEV$
1980 PRINT"Szuletesnap        ";SZNAP$
1990 PRINT
2000 REM vissza a foprogramra
2010 RETURN
```

Itt, az 1740-es sorban a program beolvassa az adattétel számát a **SZAM** változóba. Minthogy az adattételek számát sehol sem rögzítettük, a beolvasási folyamatnál előbb meg kell győződnünk arról, hogy a beírt szám megengedhető-e. Egy FOR—NEXT ciklusban (1800—1860-as sor) a program egyenként beolvas minden adattételt egészen a kívánt számig. Ha eközben az EOF(1) rendszerváltozó értéke — amely normálisan nulla — megváltozik, akkor ez azt jelenti, hogy elértük az állomány végét. (EOF(1) jelentése: End Of File és az 1-es az 1. adatátviteli csatornát jelöli.) Így az 1810-es sorban ellenőrizhetjük, hogy elértük-e az állomány végét (l. még a fejezet végi megjegyzést). Ha még nem értünk az állomány végére, a program az 1850-es sorban beolvassa a következő adattételt. Ha elérte a megadott számot, kiírja a képernyőre az adatokat a kívánt formában. 1960—1980-as sorok. Visszatérve a főprogramba, a 420-as sorban kérdezi a gép, hogy bemutasson-e még egy adattételt. Ha az adattétel száma túl nagy volt, akkor az 1830-as sorban a **HIBA** változó (jelző) értéke 1-re változik. Ebben az esetben az

Adattetel nem letezik!

Üzenet megjelenése után visszatérünk a főprogramba, és ott a 400-as sor hatására ismételten az adattétel számának kéréséhez kerülünk.

A TETEL BEMUTATASA alprogramban használtuk először azokat az új parancsokat, amelyek a számítógép és a lemezmeghajtó közötti adatforgalomra vonatkoznak. (1770-es és 1890-es sorok: ezek BASIC verzió specifikusak, tehát át kell írni, ha APPLE vagy C 64-es számítógéppel dolgozunk.

```
500 REM HOZZATETEL ----------------------------
510 OPEN DNAME$ FOR APPEND AS #1
520 CLS:PRINT:PRINT
530 PRINT"Nev                  ";:INPUT CSNEV$
540 PRINT"Utonev               ";:INPUT UNEV$
550 PRINT"Szuletesnap          ";:INPUT SZNAP$
560 PRINT #1,CSNEV$;",";UNEV$;",";SZNAP$
570 PRINT:PRINT"Hozzategyek meg egy tetelt?  (I/N)";:INPUT BEIRAS$
580 IF (BEIRAS$="i") OR (BEIRAS$="I") THEN 520
590 REM ELSE vissz a menure
600 CLOSE 1: GOTO 140
```

2. választás: HOZZATETEL

Ennél a választásnál az állományt APPEND üzemmódban nyittatjuk meg és az 530-as sortól az 560-as sorig beírjuk a nevet, az utónevet és a születésnapot a megfelelő változókba. Az adatokat az 560-as sorban a korábban ismertetettek szerint a PRINT#1 utasítással íratjuk be az állományba. Elválasztójelként vesszőket használunk.

```
650 REM torles ------------------------------
660 CLS:PRINT
670 PRINT"Melyik adattetelt toroljem?"
680 GOSUB 1700   :REM  Tetelszam be , adat ki
690 IF SZAM=0 THEN 140   :REM vissza a menure
700 REM ELSE
```

```
710 IF HIBA=1 THEN 670   :REM nem ervenyes tetelszam
720 REM ELSE adattetelt torolni
730 GOSUB 2100
740 PRINT:PRINT:PRINT
750 PRINT"Toroljek meg egy adattetelt?  (I/N)";
760 INPUT BEIRAS$
770 IF (BEIRAS$="i") OR (BEIRAS$="I") THEN 660
780 REM ELSE vissz a menure
790 GOTO 140
800 REM program vege--------------------------
810 END
```

3. választás: TORLES

Ez a programrész a 650-es sorban kezdődik és ugyanúgy mint a BEMUTATAS, ez is a TETEL BEMUTATASA alprogramot hívja meg. Ezzel kiíratja a képernyőre a törlendő az adattételt, amennyiben létezik és visszatér a főprogramba. Ezután meghívja a TETEL TORLESE alprogramot.

```
2099 :
2100 REM ***** AP: Tetel torlese *****
2110 REM a szam-ban meghatarozott tetelt torolni
2120 REM ugy,hogy az allomanyt atmasoljuk egy uj
2130 REM allomanyba (a tetel nelkul) es utana az
2140 REM uj allomanynak adjuk a regi nevet.
2150 PRINT"Igazan torolni akar?  (I/N)";
2160 INPUT BEIRAS$
2170 IF (BEIRAS$<>"i") AND (BEIRAS$<>"I") THEN 2340
2180 REM ELSE adattetelt torolni
2190 OPEN DNAME$ FOR INPUT AS #1
2200 OPEN "uj.$$$" FOR APPEND AS #2
2210 MUTATO=0
2220 IF EOF(1) THEN 2300 :REM a regi allomany vege
2230    REM ELSE adattetel be, uj allomany irasa
2240    MUTATO=MUTATO+1  :REM a regi allomany kovetkezo adatara allni
2250    INPUT #1,CSNEV$,UNEV$,SZNAP$ :REM tetel be
2260    IF SZAM=MUTATO THEN 2290 :REM nem irni
2270       REM ELSE nincs torolt tetel
2280       PRINT #2,CSNEV$;",";UNEV$;",";SZNAP$
2290    GOTO 2220 : REM kovetkezo tetel
2300 CLOSE      :REM nyitva levo allomanyokat lezarni
2310 KILL DNAME$    :REM regi allomany torlese
2320 NAME "uj.$$$" AS DNAME$
2330 PRINT:PRINT"        Adattetelt toroltuk!"
2340 RETURN
```

A véletlen törlés elkerülésére a program megkérdezi a 2150-es sorban

Igazan torolni akar? (I/N)

Ha I-t vagy i-t írunk be, a program törli az adattételt. Ehhez két megnyitott állományra van szüksége (2190-es, 2200-as sor): amelyből törölni kell (1-es csatorna) és egy segédállományra uj.$$$ néven (2-es csatorna). Ebbe az új állományba másolja be a program az összes eredeti adattételt a törlendő kivételével (2220—2290-es sor). A másolási folyamat befejeztével a program lezárja az állományokat (CLOSE), teljes egészében törli a régi állományt (2310-es sor) és 2320-as sor az új állománynak a régi nevet adja.

Ezzel a módszerrel minden törlő folyamatban az egész állományt átmásolja a program, ezért ez az eljárás meglehetősen gazdaságtalan, különösen a nagyobb állományoknál. Ennek ellenére igen szemléletesen ábrázolja az állományok kezelését.
A KILL és a RENAME parancsok megfelelői a C 64-es BASIC-ben

```
OPEN 1,8,15,”S:”+NEV$:CLOSE 1
```

illetve

```
UJ$=”UJ.$$$”
OPEN 1,8,15,”R:”+UJ$+”=”+NEV$:CLOSE 1
```

Az APPLE BASIC-ben a megfelelő utasítások:

```
PRINT D$;”OPEN”; NEV$
PRINT D$;”DELETE”;NEV$
```

illetve

```
PRINT D$;”RENAME UJ.$$$;NEV$
```

Ha a számítógépe nem fogadja el ezeket a parancsokat, nézzen utána a megfelelő nyitási, törlési és átnevezési parancsoknak a gép BASIC kézikönyvében.

Megjegyzés:

Az APPLE az állományok végének vizsgálatához (EOF) saját rendszert alkalmaz. Ez néhány olyan módosítást igényelne a programban, amelyekre itt nem tudunk kitérni.

És végül tekintse meg a teljes programlistát:

```
10 REM Adatkezeles
20 REM ----------------------------------------
30 REM Egy csaladai es utonevekbol allo szuletesi
40 REM listat akarunk vezetni.A program tegye le-
50 REM hetove adattetelek iranyitott bemutatasat,
60 REM torleset es kiegesziteset.
70 REM ----------------------------------------
99 :
100 REM beirasi kepernyok es az egyes beirasok
101 REM ====================================
110 GOSUB 1000      :REM Magyarazatok
120 INPUT BEIRAS$  :REM uj vagy meglevo allomany
130 GOSUB 1300      :REM allomanyok nevei
140 REM Menu bemutatasa es valasztas, vegrehajtas
150 GOSUB 1500      :REM menu-kepernyo
160 INPUT VALASZTAS$  :REM felhasznaloi valasztas
170 REM ugras a valasztott programreszhez
180 IF (VALASZTAS$="b") OR (VALASZTAS$="B") THEN 340
190 IF (VALASZTAS$="h") OR (VALASZTAS$="H") THEN 500
200 IF (VALASZTAS$="t") OR (VALASZTAS$="T") THEN 650
210 IF (VALASZTAS$="v") OR (VALASZTAS$="V") THEN 810
```

```
220 REM ELSE rossz beiras
230 GOTO 140          :REM vissza a Menure
299 :
300 REM FORESZ allomany lekerdezese
301 REM ===================
310 REM Ez a resz negy reszre oszlik, a negy
320 REM menupontnak megfeleleoen:
330 REM BEMUTATAS, HOZZATETEL, TORLES, VEGE.
340 REM BEMUTATAS ---------------------------
350 CLS:PRINT
360 PRINT"Melyik adattetelt mutassam be?"
370 GOSUB 1700       :REM Tetelszam be, adat ki
380 IF SZAM=O THEN 140  :REM vissza a menure
390 REM ELSE
400 IF HIBA=1 THEN 360  :REM nem ervenyes tetelszam
410 REM ELSE
420 PRINT"Bemutassak meg egy adattetelt?  (I/N)";:INPUT BEIRAS$
430 IF (BEIRAS$="i") OR (BEIRAS$="I") THEN 350
440 REM ELSE vissza a menure
450 GOTO 140
500 REM HOZZATETEL --------------------------
510 OPEN DNAME$ FOR APPEND AS #1
520 CLS:PRINT:PRINT
530 PRINT"Nev                    ";:INPUT CSNEV$
540 PRINT"Utonev                 ";:INPUT UNEV$
550 PRINT"Szuletesnap            ";:INPUT SZNAP$
560 PRINT #1,CSNEV$;",";UNEV$;",";SZNAP$
570 PRINT:PRINT"Hozzategyek meg egy tetelt?  (I/N)";:INPUT BEIRAS$
580 IF (BEIRAS$="i") OR (BEIRAS$="I") THEN 520
590 REM ELSE vissz a menure
600 CLOSE 1: GOTO 140
650 REM torles ------------------------------
660 CLS:PRINT
670 PRINT"Melyik adattetelt toroljem?"
680 GOSUB 1700    :REM  Tetelszam be , adat ki
690 IF SZAM=0 THEN 140   :REM vissza a menure
700 REM ELSE
710 IF HIBA=1 THEN 670   :REM nem ervenyes tetelszam
720 REM ELSE adattetelt torolni
730 GOSUB 2100
740 PRINT:PRINT:PRINT
750 PRINT"Toroljek meg egy adattetelt?  (I/N)";
760 INPUT BEIRAS$
770 IF (BEIRAS$="i") OR (BEIRAS$="I") THEN 660
780 REM ELSE vissz a menure
790 GOTO 140
800 REM program vege-------------------------
810 END
996 :
997 :
998 REM ************* ALPROGRAMOK *************
999 :
1000 REM ***** AP: Magyarazatok *****
1010 CLS
1020 PRINT TAB(6)"A D A T K E Z E L E S"
1030 PRINT TAB(6)"====================="
1040 PRINT:PRINT
1050 PRINT"A program lehetove teszi egy csaladi neveket, "
1060 PRINT"utoneveket es szuletesnapokat nyilvantarto    "
1070 PRINT"allomany elkesziteset es kezeleset.":PRINT
1080 PRINT"Ebben az allomanyban az egyes adatteteleket beol-"
1090 PRINT"vashatja, torolheti, avagy a meglevokhoz      "
1100 PRINT"hozzatehet."
```

```
1110 FOR I=1 TO 5:PRINT:NEXT I
1120 PRINT"   Uj allomany letrehozasa...............U"
1130 PRINT"   Meglevo allomany hasznalata...........M"
1140 FOR I=1 TO 3:PRINT:NEXT I
1150 PRINT TAB(14);"Kerem valasszon ";
1160 RETURN
1299 :
1300 REM ***** AP: Allomany megnyitasa *****
1310 REM az allomany nevenek bekerese
1320 CLS:FOR I=1 TO 5 :PRINT :NEXT I
1330 PRINT"Kerem most adja meg az allomany"
1340 PRINT"nevet.":PRINT
1350 PRINT"Allomanynev: ";:INPUT DNAME$
1360 REM ha mar megvan az allomany, semmit sem kell tenni
1370 IF (BEIRAS$="b") OR (BEIRAS$="B") THEN 1410
1380 REM ELSE uj allomany letrehozasa
1390 OPEN DNAME$ FOR APPEND AS #1
1400 CLOSE 1
1410 RETURN
1499 :
1500 REM ***** AP: Menu bemutatasa *****
1510 CLS
1520 PRINT TAB(6);"A D A T K E Z E L E S"
1530 PRINT TAB(6);"====================="
1540 FOR I=1 TO 7 :PRINT:NEXT I
1550 PRINT"Adattetelt bemutatni ..................... B"
1560 PRINT"           hozzatenni .................... H"
1570 PRINT"           torolni ....................... T"
1580 PRINT
1590 PRINT"A program VEGE ........................... V"
1600 PRINT:PRINT:PRINT
1610 PRINT"           Kerem valasszon!";
1620 RETURN
1699 :
1700 REM ***** AP: Tetel bemutatasa *****
1710 REM Tetelszam be, adat ki
1720 PRINT"Kerem adja meg a tetelszamot!"
1730 PRINT TAB(10);"( 0 ----> Vissza a menure! )"
1740 PRINT"    szam: ";:INPUT SZAM
1750 IF SZAM=O THEN 2000 : REM vissza a menure
1760 REM ELSE a kivant adattetelt megkeresni es beolvasni
1770 OPEN DNAME$ FOR INPUT AS #1
1780 HIBA=0
1790 REM addig keresni, amig a kivant tetelt meg nem talaljuk
1800 FOR I=1 TO SZAM
1810   IF EOF(1)=0 THEN 1850 : REM meg nem ertuk el az allomany veget
1820     REM ELSE allomany vege ==== hibajelzes
1830     HIBA=1
1840     I=SZAM : GOTO 1860   :REM keresest abbahagyni
1850   INPUT #1,CSNEV$,UNEV$,SZNAP$ :REM tetel be
1860 NEXT I
1870 REM  hiba=0 -----> megvan a tetel
1880 REM  hiba=1 -----> nincs meg
1890 CLOSE 1
1900 IF HIBA=0 THEN 1940 :REM ervenyes tetelszam, adat be
1910 REM ELSE elertuk az allomany veget, hibajelzes
1920 PRINT:PRINT"  Adattetel nem letezik!":PRINT
1930 GOTO 2000
1940 REM adattetel megvan, bemutatni!
1950 PRINT:PRINT
1960 PRINT"Nev                ";CSNEV$
1970 PRINT"Utonev             ";UNEV$
1980 PRINT"Szuletesnap        ";SZNAP$
```

```
1990 PRINT
2000 REM vissza a foprogramra
2010 RETURN
2099 :
2100 REM ***** AP: Tetel torlese *****
2110 REM a szam-ban meghatarozott tetelt torolni
2120 REM ugy,hogy az allomanyt atmasoljuk egy uj
2130 REM allomanyba (a tetel nelkul) es utana az
2140 REM uj allomanynak adjuk a regi nevet.
2150 PRINT"Igazan torolni akar?  (I/N)";
2160 INPUT BEIRAS$
2170 IF (BEIRAS$<>"i") AND (BEIRAS$<>"I") THEN 2340
2180 REM ELSE adattetelt torolni
2190 OPEN DNAME$ FOR INPUT AS #1
2200 OPEN "uj.$$$" FOR APPEND AS #2
2210 MUTATO=0
2220 IF EOF(1) THEN 2300 :REM a regi allomany vege
2230    REM ELSE adattetel be, uj allomany irasa
2240    MUTATO=MUTATO+1  :REM a regi allomany kovetkezo adatara allni
2250    INPUT #1,CSNEV$,UNEV$,SZNAP$ :REM tetel be
2260    IF SZAM=MUTATO THEN 2290 :REM nem irni
2270       REM ELSE nincs torolt tetel
2280       PRINT #2,CSNEV$;",";UNEV$;",";SZNAP$
2290    GOTO 2220 : REM kovetkezo tetel
2300 CLOSE      :REM nyitva levo allomanyokat lezarni
2310 KILL DNAME$    :REM regi allomany torlese
2320 NAME "uj.$$$" AS DNAME$
2330 PRINT:PRINT"      Adattetelt toroltuk!"
2340 RETURN
```

Új fogalmak a BASIC-ben:

OPEN, EOF(x), CLOSE, KILL, RENAME, PRINT#, INPUT#

12.3 MEGOLDÁS LOGO-BAN

Mielőtt megoldanánk LOGO-ban is a 12.1 alfejezetben leírt feladatot, azzal az alapvető kérdéssel kell foglalkoznunk, hogy hogyan készíthetünk LOGO nyelvben külső tárolóra állományt.

A 11. fejezetben már megismertük a programok tárolására vonatkozó parancsokat. A LOGO-ban a program tulajdonképpen egy speciális állomány. A LOGO a programot sorokból álló listaként kezeli; a sorok megint csak szavakból és jelekből álló listák. Mivel minden adatgyűjteményt listáknak — vagy listák listájának, vagy listák listájának a listájának, vagy ... — tekinthetünk, ezért a LOGO programtárolási módját használjuk adattárolásra is. Az adatállomány ugyanis alapjában véve nem más, mint egy kartotékdoboz sok kartotéklappal, vagyis adattételekből álló lista.

Az adattételeink maguk is mind listák és ezért megfelelnek egy — az imént említett — LOGO-program sorainak. Az adattételek azért listák, mert különálló szavakból — az egyes személyek családi neve, utóneve, születésnapja — állnak. "Szuletési ido" állományunk tehát olyan lista, amelynek az elemei is további

listák (adattételek). Ahhoz, hogy egy eljárást listává alakítsunk, a TEXT parancsot kell használnunk.
Ha egy listából azután ismét eljárást akarunk csinálni, akkor a DEFINE parancsot alkalmazzuk. Tehát, ha adatainkat mágneslemezen akarjuk elhelyezni, a DEFINE paranccsal eljárássá alakítjuk őket. Ezeket a 11. fejezetben leírtak alapján elraktározhatjuk. Ha ennek fordítottjára van szükségünk, vagyis be akarjuk olvasni az adatokat a tárba és ott fel akarjuk dolgozni, akkor betöltjük az elraktározott eljárást és a TEXT paranccsal újra listává alakítjuk, pl. a 7. fejezet (kocka).
Vizsgáljunk meg egy példát:

```
TO EREDMENYKIIRAS :L
 TABLAZATFEJLEC
 KIIRAS :L
END
```

eljárás a

TEXT "EREDMENYKIIRAS"

parancs hatására listává alakul át (a listák mindig szögletes zárójelben állnak):

[[:L] [TABLAZATFEJLEC] [KIIRAS :L]]

Megállapíthatjuk, hogy az eljárás minden egyes sora külön listaként jelenik meg, ezért ezek is szögletes zárójelek közé kerülnek. Az eljárás neve után álló lokális változó az első listaelemben szerepel [:L].
Ha egy eljárásnak nincs lokális változója, az első listaelem mindig üres lista [].

DEFINE "D[[] [TABLAZATFEJLEC] [KIIRAS :L]]

paranccsal a következő D nevű eljárást kapjuk:

```
TO D
 TABLAZATFEJLEC
 KIIRAS :L
END
```

Megjegyzés:

Ezekkel a parancsokkal lehetősége nyílik arra, hogy *egy LOGO programmal magát a programot* változtassa meg!
Mivel a másik két nyelvben erre nincs lehetőség, nem is foglalkozunk részletesen vele. Ennek ellenére igen vonzó lehetőségeket nyújt. Ezzel elmondtuk a lényegét a LOGO-beli adattárolásnak. Most már csak eljárásokba kell öltöztetni ezeket a parancsokat, és már meg is oldottuk a feladatot.
Egy olyan eljárásgyűjtemény, amely a 12.1-ben meghatározott feladatot megoldja, természetesen valamivel terjedelmesebb, mint az előző példáink.

```
TO ADATTAR
 ISMERTETES
 TAROLAS
 MENU
END
```

Az ADATTAR nevű főeljárás először az ISMERTETES eljárást hívja. Ez állítja elő az 1. képernyőt, a TAROLAS eljárás az állományneveket szolgáltatja, 2. képernyő, és végül a MENU állítja elő a 3. képernyőt.

```
TO ISMERTETES
 ND
 TAB 7 [ADATKEZELES] CR 1
 TAB 7 "'' U 25 "= CR 3
 PRINT [EGY CSALADI ES UTONEVBOL ALLO]
 PRINT [SZULETESNAP LISTAT AKARUNK]
 PRINT [VEZETNI.A PROGRAM TEGYE LEHETO-]
 PRINT [VE ADATTETELEK BEMUTATASAT]
 PRINT [TORLESET,KIEGESZITESET........]
 TAB 4 [UJ ALLOMANY NYITASA] TAB 32 "N CR 1
 TAB 4 [MEGLEVO ALLOMANY] TAB 32 "B CR 1
 TAB 12 [KEREM VALASSZON] S 3 MAKE "E INPUT
 MAKE "D []
END
```

```
TO TAROLAS
 MAKE "SP "FALSE
 ND CR 5
 PRINT [KEREM ADJA MEG AZ ALLOMANY NEVET] CR 1
 PRINT1 [ALLOMANYNEV?] S 1 MAKE "NEV INPUT
 IF :E = "N THEN STOP
 READ :NEV
 MAKE "D BF TEXT "ADAT
END
```

A TAROLAS eljárás először a születésnap listanevét kérdezi meg, amelyet a mágneslemezen kezelni akarunk, és ezt a **:NEV** változóban helyezi el. Ha az ISMERTETES-ben feltett kérdésre N(elemmel) válaszoltunk, a gép nem lép kapcsolatba a mágneslemezzel, hanem leállítja a TÁROLÁS eljárás futását, és visszaadja a vezérlést a fölérendelt ADATTAR eljárásnak. Az I(gen) beírása esetén azonban a READ :NEV paranccsal betölti a születésnap listát a mágneslemezről. Az adatállományt az ADAT eljárásban tároljuk. Ezt az eljárást a korábban leírtak szerint a TEXT ”ADAT paranccsal alakítjuk listává. Az így nyert D listából eltávolítjuk az első listaelemet (BF TEXT ”ADAT). Ez az első elem mindig üres lista, hiszen az ADAT eljárás sosem tartalmaz paramétereket lokális változókként. A maradéklistát a **:D** változóban helyeztük el (MAKE ”D ...). Az összes további eljárásban abból indulunk ki, hogy a születésnap lista a **:D**-ben található.

```
TO MENU
 ND
 TAB 7 [A D A T K E Z E L E S] CR 1
 TAB 7 "'' U 21 "= CR 8
 TAB 4 [A D A T MUTATAS] TAB 32 "Z CR 1
 TAB 12 [KIEGESZIT] TAB 32 "H CR 1
 TAB 12 [TOROL] TAB 32 "L CR 1
 TAB 4 [P R O G R A M] S 2 PRINT1 [VEGE] TAB 32 "E CR 4
 TAB 12 [KEREM VALASSZON!] S 3 MAKE "E INPUT
 IF :E = "Z THEN ND MUTAT
 IF :E = "H THEN KIEGESZIT
 IF :E = "L THEN ND ELTAVOLIT
 IF :E = "E THEN VEGE :D
 MENU
END
```

A MENÜ eljárás hozza létre a 3. képernyőt. A program a felhasználó választásától függően elágazik a MUTAT, a KIEGÉSZIT, az ELTÁVOLIT vagy a VÉGE eljárások valamelyikéhez.

Nézzük meg először a MUTAT eljárást:

```
TO MUTAT
 CR 1
 PRINT [MELYIK ADATTETELT MUTASSAM?]
 PRINT [KEREM A TETEL SZAMAT:]
 TAB 10 [(0--->VISSZA A MENUBE)] CR 1
 TAB 4 [SZAMA:] S 1 MAKE "N INPUT
 IF :N < 1 THEN MENU
 IF :N > COUNT :D THEN CR 1 PRINT1 [ADATTETEL NEM LETEZIK!] MUTAT
 FELVAG :N KESZLET :H
 PRINT1 [MEG EGY ADATTETEL(I/N)] S 1
 MAKE "E INPUT
 TEST :E = "I
 IFT MAKE "A [] MAKE "B :D ND MUTAT
 IFF MENU
END
```

Először a kívánt adattétel számát kell megadnunk. Ha 1-nél kisebb számot írunk be, a program visszalép a menühöz. Ha a beírás érvénytelen volt (:N > COUNT :D), ezt a gép kiírja a képernyőre és újra meghívja a MUTAT eljárást. Érvényes beírás esetén a KESZLET eljárás a FELVAG eljárással elérhetővé tett adattételeket a kívánt formában a képernyőre írja. Ha még egy adattételt meg akarunk nézni, újra elő kell hívni a MUTAT eljárást, különben a program a menühöz ágazik el.
Például, a hetedik adattételre vagyunk kíváncsiak (:N-ben a kívánt adattétel száma, 7 áll), akkor a KESZLET eljárásnak majd a **:D** adatlista hetedik elemét kell bemutatnia. Ahhoz, hogy az adatlistából tetszőleges adattételt kiválasszunk, a FELVAG eljárást és APRÍT nevű aleljárását használjuk.

```
TO FELVAG :N
 MAKE "A []
 MAKE "B :D
 APRIT :N
END
```

Először a FELVAG eljárás **:A** és **:B** változójában elhelyezzük az üres, illetve a teljes adatlistát, majd előhívjuk az APRIT aleljárást.

```
TO APRIT :X
 IF :X = 0 THEN STOP
 MAKE "H FIRST :B
 MAKE "A LPUT :H :A
 MAKE "B BF :B
 APRIT :X - 1
END
```

Mivel a **:D** változóban minden adattétel egyben külön lista, a **:D** teljes listát ott úgy bontjuk szét két részlistára, hogy az első részlista tartalmazza az összes adattételt, utolsóként a keresett tételt, a második részlista pedig a többit, a maradékot. Ezt a két részlistát tartalmazza az **:A** és a **:B** változó. Az APRIT a kívánt felbontást úgy hajtja végre, hogy a keresett adattétel végül a **:H**-ba kerül. Hogyan dolgozik az APRIT eljárás? Ha adattételszámként nullát írtunk be, visszaküld minket a menübe. Egyébként legalább egyszer végigfut az eljárás. Amikor belépünk az eljárásba a **:B** változó tartalmazza a teljes adatlistát (:D) (l. FELVAG eljárás). Az első átfutás végén **:H**-ba kerül az első adattétel, de még **:A**-ban is meglesz, ugyanis az LPUT a beírásokból állít össze egy listát. Az első beírás ez esetben **:H,** a második pedig **:A,** amely kezdetben csak az üres [] listát tartalmazta. Olyan listát kapunk, amelynek utolsó eleme **:H.** (Bár :H a beírások között az első helyen áll, mégis a lista végére kerül!) A **:B**-ből a BUTFIRST (BF) utasítással elvesszük az első adattételt. A folyamatot megismételve **:H**-ban megjelenik a második adattétel, **:A**-ban az első kettő és **:B**-ben pedig a maradék lista. Minden átfutásnál: **:B** első adattétele átkerül **:A**-ba és közben átmenetileg **:H**-ban is tárolódik. Mivel az APRIT eljárást minden alkalommal 1-gyel kisebb beírt paraméterrel hívjuk elő, **:N** átfutás után pontosan az n-edik adattétel lesz az **:A** lista végén, és **:H**-ban.

```
TO KESZLET :H
 CR 2
 PRINT1 [CSALADI NEV] TAB 19 FIRST :H CR 1
 PRINT1 [UTONEV] TAB 19 FIRST BF :H CR 1
 PRINT1 [SZULETESNAP] TAB 19 FIRST BF BF :H CR 2
END
```

A KESZLET eljárás ezek után már csak annyit tesz, hogy a **:H**-val átadott adattételt NEV (FIRST), UTONEV (FIRST BF) SZULETESNAP (FIRST BF BF) sorrendben kiíratja a képernyőre.

Most pedig nézzük meg a menü második pontját, hogyan egészítjük ki a születésnap listánkat újabb adattétellel?

Ez a KIEGESZIT eljárás dolga:

```
TO KIEGESZIT
 MAKE "TAROLNI "TRUE
 ND CR 2
 PRINT1 [CSALADI NEV] TAB 18 "´´ MAKE "N INPUT
 PRINT1 [UTONEV] TAB 18 "´´ MAKE "V INPUT
 PRINT1 [SZULETESNAP] TAB 18 "´´ MAKE "G INPUT
 MAKE "S ( LIST :N :V :G )
 MAKE "D ( SENTENCE :D ( LIST :S ) ) CR 1
 PRINT1 [MEG EGY KIEGESZITES(I/N)?]
 MAKE "E INPUT
 TEST :E = "I
 IFT KIEGESZIT
 IFF MENU
END
```

A **:TAROLNI** változót IGAZ-ra (”TRUE) állítjuk, hogy a főprogram befejezésekor felismerhető legyen, megváltozott-e a születésnap lista. A KIEGESZIT eljárás következő részében az adatbeviteli forma épül fel, és a program minden alkalommal kéri a hozzá tartozó adatot. Ezekből alakítja ki azután az **:S** listát, amelyet a SENTENCE LOGO-szóval fűz a már meglévő **:D** listához. A SENTENCE változó számú szavakat vagy listákat kapcsol össze egy új listává. Példánkban a meglévő születésnap listát (:D) és a már kialakított **:S** listát (SENTENCE :D (LIST :S)) kapcsolja össze az új **:D** listává (MAKE ”D ...). Végül megkérdezi a program, hogy kell-e

MEG EGY KIEGESZITES (I/N)?

és a választól függően vagy újra hívja a KIEGESZIT eljárást, vagy visszamegy a menü-be.

A menü harmadik pontja egy adattétel törlése. Ezt az ELTAVOLIT eljárás intézi:

```
TO ELTAVOLIT
 CR 1
 PRINT [MELYIK ADATTETELT TOROLJEM?]
 PRINT [KEREM ADJA MEG A SZAMAT!]
 TAB 10 [(0-->VISSZA A MENUHOZ)] CR 1
 TAB 4 [A SZAM:] S 1 MAKE "N INPUT
 TEST :N < 1
 IFT MENU
 IFF TEST :N > COUNT :D
 IFT CR 1 PRINT1 [ADATTETEL NEM LETEZIK] ELTAVOLIT
 IFF TOROL
 PRINT1 [MEG EGY ADATTETELT TOROLJEK?(I/N)] S 1 MAKE "E INPUT TEST :E = "I
 IFT MAKE "A [] MAKE "B :D ND ELTAVOLIT
 IFF MENU
END
```

Az eljárás szerkezete a MUTAT-hoz teljesen hasonlóan épül fel. Először ez is a törlendő adattételt kérdezi meg. Amennyiben a beírt tétel nem létezik, hibajelzést kapunk, ill. ha nullát írunk be az adattétel számaként, akkor az eljárás visszatér a menühöz. Ha a beírt tétel létezik, akkor a TÖRÖL eljárást hívja meg, és az törli a kívánt tételt. Ezután ismét egy kérdés következik, hogy töröljön-e még egy adattételt, majd a választól függően magát (ELTAVOLIT) vagy a MENU eljárást hívja meg. Egy adattétel törlésénél az ELTAVOLIT eljárás csak irányítja a folyamatot, a tényleges munkát a TOROL eljárás végzi:

```
TO TOROL
 FELVAG :N
 KESZLET :H
 PRINT1 [BIZTOS ON EBBEN?(I/N)] S 1 MAKE "E INPUT CR 1
 IF :E = "N THEN CR 2 STOP
 MAKE "TAROLNI "TRUE
 MAKE "A BL :A
 MAKE "D SENTENCE :A :B
 TAB 7 [ADATTETEL TOROLVE!] CR 4
END
```

Az eljárás első részét a kommentárok magától értetődővé teszik. Csak a tényleges törlésnél válik a helyzet bonyolultabbá. Hogy a **:TAROLNI** miért kap igaz (TRUE) értéket, azt már a KIEGESZIT eljárásnál elmondtuk. A FELVAG eljárás hívása után, ahogy ezt már az APRIT eljárásnál tisztáztuk, a törlendő adattétel az **:A**-ban található a lista utolsó elemeként. A **MAKE "A BL :A** utasítással kimetsszük ezt az elemet **:A**-ból. Majd az új **:A** listából és a változatlanul maradt **:B**-ből elkészítjük az új **:D** születésnap listát.
A törléskor minden adattétel sorszáma megváltozik a törölt adat után. A számozást úgy tarthatnánk meg, ha a törölt adatok helyét üres listákkal tölthetnénk fel. Ez viszont egyrészt gondokat okozna a kiíratásnál, másrészt fölöslegesen foglalná a tárolóhelyet. Most már csak az utolsó opció, a VÉGE hiányzik a műveletből. Ezt a VÉGE eljárás valósítja meg:

```
TO VEGE
 IF :TAROLNI = "FALSE THEN CR 3 TOPLEVEL
 DEFINE "ADAT FPUT [] :D
 ( SAVE :NEV [ADAT] )
 CR 3 TOPLEVEL
END
```

A TAROLAS eljárásban a **:TAROLNI** változó hamis (FALSE) értéket kapott. Ha csak olvassuk az adattételeket, akkor változatlan marad, hiszen csak a KIEGESZIT és a TOROLNI eljárásokban kap a **:TAROLNI** változó igaz (TRUE) értéket. Tehát ez a változó jelzi a VEGE eljárásnak, hogy változott-e az adatállomány. Így a **:TAROLNI** értékétől függ, hogy a lemezre íráshoz át kell-e alakítani a listát eljárássá. Amennyiben igaz értékű, először az első helyre egy üres listát kell hozzáírni és csak azután alakítjuk át az ADAT eljárássá (DEFINE "ADAT ...). Ezt az eljárást azután a **:NEV** változóban tárolt állománynév alatt visszük ki a mágneslemezre. Megfigyelhetjük, hogy a SAVE-nél nem kell közvetlenül beírnunk az állomány nevét, hanem megadhatjuk egy változóval is. Hogy ne írjuk ki a lemezre ezzel együtt a teljes tártartalmat, kiterjeszthetjük a SAVE parancsot: az állomány neve mögött szögletes zárójelben listaként tüntethetjük fel azoknak az eljárásoknak a neveit, amelyeket lemezre akarunk menteni. Az esetünkben ennek a listának csak egyetlen eleme van, az ADAT eljárás. A kerek zárójel biztosítja, hogy a lemezre mentés csak a lista elemeire korlátozódjon. Amint az adatok a lemezre kerültek, a program a TOPLEVEL-lel befejeződik.

Új fogalmak a LOGO-ban:

LPUT, SENTENCE, TEXT, DEFINE, SAVE kiterjesztése, eljárás átalakítása listává és viszont

12.4 PASCAL PROGRAMOZÁS

Három programunk közül a Pascal a leghosszabb. A Pascalban minden lehetőség megvan egy kiváló teljesítményű állománykezelő program kialakításához. Ebből következik, hogy egyszerű állományokra vonatkozó feladatok esetén is célszerű egy későbbi, kiegészítő programszerkesztés lehetőségét szem előtt tartani. A másik két nyelvvel ellentétben a Pascalban a RANDOM adattárolásnak (véletlenállomány) nevezett módszert alkalmazzuk. Az ilyen állományt szokták közvetlen elérésű állománynak is nevezni. Erre a típusra az jellemző, hogy az adatállomány egyenlő hosszúságú elemekre oszlik, ezek az elemek számozottak, és a szám megadásával közvetlenül kiválaszthatók. Számolni kell azzal, hogy a lemezen sok tárolóhely kárba vész, mert az elemek nagyságát a mezőikben lehetséges leghosszabb beírás határozza meg.

Ha a BASIC-ben állományra hivatkozunk, akkor a csatornaszámot használjuk. A Pascal filozófiája is hasonló, de a csatornaszámokat jelölésekkel helyettesítjük: ezek az *állományváltozók,* vagy egyszerűen *állományok.* Ezeket a jelöléseket a program deklarációs részében kell elhelyeznünk:

```
VAR csatorna 1 : FILE OF integer;
    csatorna 2 : FILE OF char;
```

Azt is meg kell adni, hogy milyen típusú adatokat akarunk tárolni. Az előző két példában az egyik egész típusú, a másik karakter típusú. Esetünkben családi neveket, utóneveket, születésnapokat akarunk adatállományba juttatni, vagyis STRING adatokat. Ismét kihasználjuk, hogy a Pascalban saját adattípusokat definiálhatunk és meghatározunk magunknak egy olyan adattípust, amely egy kartotéklapot ábrázol:

```
TYPE szemely = RECORD
           név : STRING [20];
        utónév : STRING [20];
   születésnap : STRING [10];
END;
```

A RECORD olyan adattípus, amely különböző adattípusok összekapcsolódását fejezi ki. A kapcsolódáson belül az egyes adattípusok tetszőlegesek lehetnek, egymástól különbözhetnek — vagyis nem kell azonos típusúaknak lenniük, mint az eddigi feladatainkban. Ha egy RECORD valamelyik mezőjével kapcsolatba akarunk kerülni, pontot (.) kell tenni a recordváltozó neve után és emögött kell feltüntetnünk a mező megjelölését:

```
szemely.utonev: = 'Jozsef';
```

Az adatállomány olyan összetevők sorozata, amelyek mind ugyanahhoz az adattípushoz tartoznak. Ezt a típust *összetevő* vagy *komponens* típusnak hívják, és tetszés szerint lehet kérni. Mivel a személlyel a feladatunknak megfelelő adattípust definiáltunk, az alábbi típust határozhatjuk meg adattípusként állomány változónk számára:

TYPE kartotek := FILE Of szemely;

Ezzel eldöntöttük, hogy a lemezre írt adattömbök személy típusúak lesznek.

Ahogy a BASIC-ben, úgy a Pascalban is kapcsolatot kell létesítenünk a programon belüli állománynév és a mágneslemezen szereplő állománynév között. Ehhez deklarálunk egy kartotéknév változót:

VAR kartoteknev : STRING [14];

A program futása során valamikor ez a változó majd értéket kap. Ha ugyanilyen nevű állományt akarunk a mágneslemezen használni, akkor az ASSIGN (kartotek, kartotéknev); utasítással teremtjük meg a kapcsolatot. Ha az állomány még nem áll rendelkezésre a mágneslemezen, akkor ezután a REWRITE (kartotek) parancsnak kell következnie. Ez a parancs az állományt írásra készíti elő (megnyitja). Létező állomány esetén pedig töröli a tartalmát, és az elejére állítja az adattétel-mutatót. Most már dolgozhatunk az állománnyal: WRITE (szamok, x) beírja az állományba **x** tartalmát, ha a **szamok**-at FILE OF integerként és **x**-et integer típusú változóként deklaráltuk.
Ha olvasni akarunk a lemezről, akkor az ASSIGN (..., ...) utasítással jelentkeztetjük be az állományt és a RESET (kartotek)-kal készítjük elő az olvasásra. Eközben az adattétel-mutató az elejére áll, de az állomány tartalma nem törlődik. Ilyenkor az állománynak már rajta kell lennie a lemezen. A READ (kartotek, ...) utasítással olvashatjuk be a lemezről az adatokat.
Ha befejeztük a munkát az állománnyal, a CLOSE (kartotek) utasítással le kell zárnunk az állományt.

Ezután az előzetes magyarázat után nekiláthatunk a Pascal program elemzésének. Tárgyalását a deklarációs résszel és a főprogrammal kezdjük. Az eljárásokról, amelyek a Pascal programban a főprogram előtt állnak, a jobb áttekinthetőség miatt majd csak a végén ejtünk szót részletesebben.

```
program szuletesnapok;
type    szemely = record
           csaladinev   :string[20];
           utonev       :string[20];
           szuletesnap  :string[8];
        end;
        adattar   =file of szemely;
var  beiras, valasztas          :char;
     tovabb                     :boolean;
     valtoztatott               :boolean;
     kartoteknev                :string[14];
     kartoteklap                :szemely;
     lapok_nr                   :integer;
     tetelszam                  :integer;
     kartotek                   :adattar;
```

A program a szuletesnapok nevet kapta. A TYPE-pal definiált adattípusokat már tárgyaltuk.

A **beiras** és a **valasztas** változóba írja be a felhasználó, hogy régi vagy új állománnyal akar-e dolgozni, illetve milyen feldolgzást akar végezni. Ezek tehát csak karakteres változók lehetnek.

A **tovabb** és **valtoztatott** két jelzőváltozó. A **tovabb** a program végére van hatással, a **valtoztatott** pedig azt jelzi, hogy változtattunk-e az állományban valamit, kell-e lemezre menteni az állományt, avagy sem.

A **kartoteknev** olyan változó, amely átveszi az állománynevet a lemezről. Ezért eleget kell tennie a CP/M és az MS—DOS előírásainak, amelyek maximálisan 14 karakter hosszú neveket engedélyeznek.

A **kartoteklap** — személy típusú változó, a kartotéklapba olvashatjuk be a személyek adatait. Az állományok „kartotékdoboz" modelljében a kartotéklap egy üres vagy kitöltött cédulának felel meg.

A **lapok_nr** mutató, az egyes adattételek megjelölésére használjuk.

A **tetelszam** felöleli a rendelkezésre álló adattételek összességét.

A **kartotek** a kitöltött kartotéklapok összességét foglalja magába, megfelel a LOGO megoldás **:D** változójának.

A főprogram áttekintésekor vegye figyelembe azokat az állományfunkciókat, amelyeket a 12.1 alfejezetben a képernyőkiírásokkal vezettünk be. A kommentárokkal együtt így már sok minden világos lesz:

```
begin { F O P R O G R A M }
  ismertetes;
  keszlet;
  valtoztatott:=false;
  tovabb:=false;
  repeat  {addig, amig tovabb=true }
      menu;
      case valasztas of
         'b','B':{ adattetelt bemutatni}
                 begin
                    assign(kartotek,kartoteknev);reset(kartotek);
                    tetelszam:=filesize(kartotek);
                    repeat  { amig 0-t nem irunk be }
                        clrscr;writeln;
                        tetelbemutatas('b');
                        if lapok_nr>0 then
                            begin
                               writeln;
                               write('Ker meg egy tetelt?...(I/N)');
                               readln(beiras);
                            end
                    until (lapok_nr=0) or not (beiras in ['i','I']);
                    close(kartotek);
                 end;
         'h','H':{ adatokat beirni }
                 begin
                    assign(kartotek,kartoteknev);reset(kartotek);
                    seek(kartotek,filesize(kartotek));
                    repeat
                       iras;
                       writeln;
                       write('Hozzategyek meg egy tetelt?...(I/N)');
                       readln(beiras);
                    until not (beiras in ['i','I']);
                    close(kartotek);
                 end;
         't','T':{ adatok torlese }
                 begin
                    assign(kartotek,kartoteknev);reset(kartotek);
                    tetelszam:=filesize(kartotek);
                    repeat { ismetelni, amig 0-t irunk be }
                       clrscr; writeln;
                       tetelbemutatas ('t');
                       if lapok_nr >0 then

                            begin
                               writeln;
                               write('Igazan torolni akar?..(I/N)');
                               readln(beiras);writeln;
                               if beiras in ['i','I'] then
                                  begin
                                    torles(lapok_nr);
                                    writeln(' adattetelt toroltem!');
                                    writeln;
                                  end;
                               writeln;writeln;
                               write('Toroljek meg egy tetelt?..(I/N)');
                               readln(beiras);
                            end
                    until (lapok_nr=0) or not (beiras in ['i','I']);
                    close(kartotek);
                 end;
         'v','V':{Program Vege!}
                 begin
                    vege;
                    tovabb:=true
                 end;
      end
  until tovabb;
end.
```

A főprogram az ISMERTETES eljárás előhívásával kezdődik. Ez íratja ki a bevezető tájékoztatásokat az 1. képernyőre.

```
procedure ismertetes;
  var i:integer;
  begin
     repeat  { amig helyes a beiras }
        clrscr;
        writeln('    A D A T K E Z E L E S     ');
        writeln('    =====================     ');
        writeln;writeln;
        writeln('A program lehetove teszi  csaladi neveket,   ');
        writeln('utoneveket es szuletesnapokat nyilvantarto   ');
        writeln('adattar elkesziteset es kezeleset.           ');
        writeln;
        writeln('Ebben az adattarban egyes adatteteleket iranyi-');
        writeln('tottan leolvashat, a meglevokhoz hozzatehet, ');
        writeln('illetve torolhet.');
        for i:=1 to 5 do writeln;
        writeln('Uj adattarat akar keszitení................U  ');
        writeln('Mar meglevot hasznal.......................M  ');
        for i:=1 to 3 do writeln;
        write('              Kerem, valasszon! ');
        readln(beiras)
    until beiras in ['u','U','m','M']
 end;
```

A KESZLET eljárásban nevezzük meg az állományt és az ISMERTETES eljárásbeli választól függően törölhetjük, újraírhatjuk (REWRITE(kartotek)) az állományt.

```
procedure keszlet;
{ A felhasznalo itt irja be a kivant adattarnevet.Ha }
{ uj adattarat akar keszíteni, ezen a neven keszul el }
{ az uj adattar.                                        }
 var i:integer;
 begin
    clrscr;
    for i:=1 to 5 do writeln;
    writeln('Kerem most irja be az ');
    writeln('adattar nevet.');writeln;
    write('adattar neve?');readln(kartoteknev);
    if beiras in ['u','U'] then { uj adattar keszitese }
       begin
           assign(kartotek,kartoteknev); { a valasztott nevet hozza- }
                                         { rendeli az adattarhoz,  }
           rewrite(kartotek);            { elkesziti az allomanyt, }
                                         {  es ujra lezarja.       }
           close(kartotek);
       end;
   {ELSE a felhasznalo nem akar uj allomanyt letrehozni!           }
end;
```

Ezután a jelzőváltozóknak adunk kezdőértéket.
Az összes további utasítás a REPEAT—UNTIL **tovabb** ciklusban található és a ciklus addig fut, amíg a jelzőváltozó értéke igaz (TRUE) nem lesz.
A főprogram, mint általában, most is egy sor eljárást használ.
A MENU eljárásban a gép beírást vár a felhasználótól.

```
procedure menu;
 var i : integer;
 begin
    repeat { amig helyes beiras van }
        clrscr;
        writeln('       A D A T K E Z E L E S         ');
        writeln('       =====================         ');
        for i:=1  to 7 do writeln;
        writeln('Adattetelt bemutatni.............B');
        writeln('           hozzatenni............H');
        writeln('           torolni...............T');
        writeln;
        writeln('A Program Vege...................V');
        for i:=1 to 3 do writeln;
        write('Kerem, valasszon!');
        readln(valasztas);
    until valasztas in ['b','B','h','H','t','T','v','V'];
 end;
```

A választást követően a **valasztas** változóba kerül a felhasználó kívánsága (b, h, t vagy v). Ezt a változót a gép a CASE ... OF-fal kérdezi meg. A beírt betűtől függően a Pascal végrehajtja a megfelelő programrészt. Hogy elkerüljük a kis- vagy nagybetűs beírás okozta problémát, a program mindenütt elfogadja mindkét lehetőséget.

Kezdjük az elemzést a b választáshoz tartozó részprogrammal. Ez a részprogram is, mint a többi, BEGIN és END között áll. Ahogy ezt már korábban leírtuk, az ASSIGN-nal kapcsoljuk az állományt az állományváltozóhoz és a RESET-tel a kezdést az első adattételre állítjuk. A FILESIZE TURBO—Pascal függvénnyel meghatározzuk az adattételek számát. A REPEAT ciklusban először a képernyőt töröljük, majd a TETELBEMUTATAS eljárást hívjuk.

```
procedure tetelbemutatas (valasztas:char);
 { Ez az eljaras keszíti el a bemutatashoz, illetve  }
 { a torleshez szukseges kepernyot.                  }
 begin
    repeat { amig ervenyes tetelszam van }
        if valasztas='t'
            then writeln('Melyik tetelt toroljem?')
            else writeln('Melyik tetelt mutassam?');
        writeln('Kerem adja meg a tetelszamot!');
        writeln('( 0 ----> Vissza a menure.)':26);
        write('szam: ':10);readln(lapok_nr);
        if lapok_nr>tetelszam then
            begin  { Hibajelzes }
                writeln;
                writeln('     adattetel nem letezik! ');
                writeln
            end
    until lapok_nr<=tetelszam;
    if lapok_nr>0
        then  { ervenyes kartoteklap }
            lap (lapok_nr)
        {ELSE  beiras 0 volt }
 end;   { vissza az elohivo programba }
```

A TETELBEMUTATAS az az eljárás, amely mind bemutatáskor, mind törléskor felépíti a képernyőt, és rossz adattételszám esetén hibajelzést ad.
A ciklust kétféleképpen hagyhatjuk el: nullát írunk be a kívánt adattétel számaként vagy N(em)-mel válaszolunk az alábbi kérdésre:

Ker meg egy tetelt? ... (I/N)

A **beiras IN ['i','I']** eredménye igaz (TRUE), ha a **beírás** i vagy I értéket kapott, egyébként hamis (FALSE). A NOT operátor negálja a kapott igazságértéket.
Maradjunk még mindig a főprogram elemzésénél! Megállapíthatjuk, hogy a két aktív részprogram, a kiegészítés és törlés, felépítése nagyjából azonos.
A kiegészítésnél a SEEK függvényt használjuk, hogy az adattételek számával, amelyet a FILESIZE függvénnyel állapítunk meg, az állomány végére ugorhassunk, s ott fűzzük hozzá az új adattételt.

```
begin
   seek(kartotek,szam-1);
   kartoteklap.csaladinev:='*';
   write(kartotek,kartoteklap);
   valtoztatott:=true;
end;
```

A hozzáírást az IRAS eljárással végezzük.

```
procedure lap (szam:integer);
 begin
    writeln;writeln;
    seek(kartotek,szam-1);          { lapot megkeresni }
    read(kartotek,kartoteklap);     { elolvasni        }
    with kartoteklap do             { bemutatni        }
        begin
           writeln('Nev                    ',csaladinev);
           writeln('Utonev                 ',utonev);
           writeln('Szuletesnap            ',szuletesnap);
        end;
 end;
procedure iras;
 begin
    clrscr;writeln;writeln;
    with kartoteklap do
        begin
           write('Nev                     ');readln(csaladinev);
           write('Utonev                  ');readln(utonev);
           write('Szuletesnap             ');readln(szuletesnap);
        end;
    write(kartotek,kartoteklap);      { adattetelt beirni }
 end;
```

A törlésbe beépítettünk egy biztonsági kérdést:

Igazan torolni akar? ... (I/N)

Előzőleg természetesen látnunk kell az illető adattételt. Ezt a TETELBEMUTATAS eljárás teszi lehetővé, amelyet már az első részprogramnál bemutattunk. Az adattételt, amelyet törölni kívánunk, a TORLES eljárás jelöli meg egy csillaggal a **csaladinev** mezőben, és ha főprogramban a VEGE eljárást hívjuk, akkor ez gondoskodik arról, hogy a törlésre kijelölt összes adattételt töröljük az adathordozóról.

Eddigi megjegyzéseinkkel a főprogram elemzését még korántsem fejeztük be. Csak a legfontosabb pontokat említettük meg. A részletes vizsgálatot Önre kell bíznunk, ugyanis ennek leírásával inkább csak összezavarnánk, mintsem segítséget nyújtanánk. Azt tanácsoljuk, hogy a program feldolgozása, megértése után, próbálja meg megváltoztatni valamelyik programfunkciót. A hozzá tartozó programrész ellenőrzésekor kitűnnek a részletek. Eleinte elég csak olyan változtatásokat végrehajtani, amelyek egyetlen eljárást érintenek a programból, a többit hagyja változatlanul.

A továbbiakban a programban használt eljárásokkal foglalkozunk. Ha működőképes programot akar készíteni, akkor ezeket a változók deklarálása és a főprogram közé kell elhelyeznie.
Az eljárásokat úgy elemezzük, hogy egyet kiválasztunk közülük és azt beszéljük meg. A VEGE eljárást választottuk, mert ebben különösen sok állománykezelés található. A többi eljárás kevesebb újdonságot tartalmaz.

```
procedure vege;
   { a feldolgozas kozben toroltkent megjelolt teteleket }
   { fizikailag torolnunk kell!                          }
 var uj_kartotek : adattar;
     lap         : szemely;
     begin
        if not valtoztatott
            then { nem volt mozgas }
            else
                begin
                    assign(uj_kartotek,'uj');
                    rewrite(uj_kartotek );
                    assign(kartotek,kartoteknev);
                    reset(kartotek);
                    while not eof(kartotek) do
                        begin
                           read(kartotek,lap);
                           if lap.csaladinev<>'*' then
                               { ezt nem toroltuk,}
                               { tehat tegyuk at! }
                               write(uj_kartotek,lap);
                        end;
                        close(kartotek);
                        erase(kartotek);
                        close(uj_kartotek);
                        rename(uj_kartotek,kartoteknev);
                    end
        end;
```

A TORLES eljárásban valójában nem töröltük a kiválasztott adattételt. Ehelyett a nevet csillaggal (*) helyettesítettük, majd az így megváltoztatott adatot visszaírattuk az eredeti helyére, a lemezre. Ez lehetséges volt, mert szemben a soros (szekvenciális) állománnyal, itt az összes adattétel egyenlő nagyságú.

A VÉGE eljárásban először azt vizsgáljuk, hogy a **valtoztatott** nevű logikai változó igaz (TRUE) értékű-e. Ez akkor van így, ha a program futása során az adattárat kiegészítettük, vagy legalább egy adat törlését előkészítettük. Tehát, ha csak megnéztük az adatokat, a **valtoztatott** értéke hamis (FALSE), ezért a NOT változtatott értéke igaz (TRUE). Ekkor a program a THEN ágra lép, ahol nincs semmi — csak egy kommentár. Ilyenkor az eljárás azonnal befejeződik. Egyébként (ELSE) a gép elkészíti az új adattárt, majd megnyitja a régi állományt is, és az adattétel-mutatót az állomány elejére állítja (RESET (kartotek);). Ezután, egészen a régi állomány végéig — EOF(kartoték): ez jelzi a kartotékadattár végét — a program minden adatot megvizsgál, hogy a **családinév** mező tartalma csillag-e. Ha az, akkor nem másolja át az új adatállományba, ha nem, akkor átírja.

A ciklus befejeztével az összes érvényes adat az új adatállományba került. A program lezárja, majd törli a régit. Lezárja az újat is, és átnevezi az eredeti állomány nevére.

És most, hogy már minden új ismeret a helyére került, íme a teljes program:

```
program szuletesnapok;
type    szemely = record
          csaladinev   :string[20];
          utonev       :string[20];
          szuletesnap  :string[8];
        end;
        adattar   =file of szemely;
var  beiras, valasztas          :char;
     tovabb                     :boolean;
     valtoztatott               :boolean;
     kartoteknev                :string[14];
     kartoteklap                :szemely;
     lapok_nr                   :integer;
     tetelszam                  :integer;
     kartotek                   :adattar;
procedure ismertetes;
  var i:integer;
  begin
     repeat  { amig helyes a beiras }
        clrscr;
        writeln('    A D A T K E Z E L E S     ');
        writeln('    =====================     ');
        writeln;writeln;
        writeln('A program lehetove teszi  csaladi neveket,   ');
        writeln('utoneveket es szuletesnapokat nyilvantarto   ');
        writeln('adattar elkesziteset es kezeleset.           ');
        writeln;
        writeln('Ebben az adattarban egyes adatteteleket iranyi-');
        writeln('tottan leolvashat, a meglevokhoz hozzatehet, ');
        writeln('illetve torolhet.');
        for i:=1 to 5 do writeln;
        writeln('Uj adattarat akar kesziteni...............U  ');
        writeln('Mar meglevot hasznal......................M  ');
        for i:=1 to 3 do writeln;
        write('          Kerem, valasszon! ');
        readln(beiras)
    until beiras in ['u','U','m','M']
 end;
procedure keszlet;
{ A felhasznalo itt irja be a kivant adattarnevet.Ha }
{ uj adattarat akar kesziteni, ezen a neven keszul el }
{ az uj adattar.                                      }
 var i:integer;
 begin
    clrscr;
    for i:=1 to 5 do writeln;
    writeln('Kerem most irja be az ');
    writeln('adattar nevet.');writeln;
    write('adattar neve?');readln(kartoteknev);
    if beiras in ['u','U'] then { uj adattar keszitese }
       begin
           assign(kartotek,kartoteknev); { a valasztott nevet hozza- }
                                         { rendeli az adattarhoz,  }
           rewrite(kartotek);            { elkesziti az allomanyt, }
                                         {  es ujra lezarja.       }
           close(kartotek);
       end;
    {ELSE a felhasznalo nem akar uj allomanyt letrehozni!        }
 end;
procedure menu;
 var i : integer;
 begin
    repeat { amig helyes beiras van }
        clrscr;
        writeln('       A D A T K E Z E L E S       ');
        writeln('       =====================       ');
        for i:=1  to 7 do writeln;
        writeln('Adattetelt bemutatni.............B');
        writeln('           hozzatenni............H');
        writeln('           torolni...............T');
        writeln;
        writeln('A Program Vege...................V');
        for i:=1 to 3 do writeln;
        write('Kerem, valasszon!');
        readln(valasztas);
    until valasztas in ['b','B','h','H','t','T','v','V'];
 end;
procedure lap (szam:integer);
 begin
    writeln;writeln;
    seek(kartotek,szam-1);       { lapot megkeresni }
    read(kartotek,kartoteklap);  { elolvasni        }
    with kartoteklap do          { bemutatni        }
        begin
            writeln('Nev                     ',csaladinev);
            writeln('Utonev                  ',utonev);
            writeln('Szuletesnap             ',szuletesnap);
        end;
 end;
procedure iras;
 begin
    clrscr;writeln;writeln;
    with kartoteklap do
        begin
            write('Nev                       ');readln(csaladinev);
            write('Utonev                    ');readln(utonev);
            write('Szuletesnap               ');readln(szuletesnap);
        end;
    write(kartotek,kartoteklap);    { adattetelt beirni }
 end;
procedure torles(szam:integer);
 begin
    seek(kartotek,szam-1);
    kartoteklap.csaladinev:='*';
    write(kartotek,kartoteklap);
    valtoztatott:=true;
 end;
procedure tetelbemutatas (valasztas:char);
 { Ez az eljaras kesziti el a bemutatashoz, illetve  }
 { a torleshez szukseges kepernyot.                  }
 begin
    repeat { amig ervenyes tetelszam van }
        if valasztas='t'
            then writeln('Melyik tetelt toroljem?')
            else writeln('Melyik tetelt mutassam?');
        writeln('Kerem adja meg a tetelszamot!');
        writeln('( 0 ----> Vissza a menure.)':26);
        write('szam: ':10);readln(lapok_nr);
        if lapok_nr>tetelszam then
            begin  { Hibajelzes }
                writeln;
                writeln('     adattetel nem letezik! ');
                writeln
            end
    until lapok_nr<=tetelszam;
    if lapok_nr>0
        then    { ervenyes kartoteklap }
```

```
            lap (lapok_nr)
        {ELSE  beiras 0 volt }
 end;   { vissza az elohivo programba }
procedure vege;
   { a feldolgozas kozben toroltkent megjelolt teteleket }
   { fizikailag torolnunk kell!                          }
 var uj_kartotek : adattar;
     lap         : szemely;
     begin
        if not valtoztatott
            then { nem volt mozgas }
            else
                begin
                    assign(uj_kartotek,'uj');
                    rewrite(uj_kartotek );
                    assign(kartotek,kartoteknev);
                    reset(kartotek);
                    while not eof(kartotek) do
                        begin
                           read(kartotek,lap);
                           if lap.csaladinev<>'*' then
                               { ezt nem toroltuk,}
                               { tehat tegyuk at! }
                               write(uj_kartotek,lap);
                        end;
                  close(kartotek);
                  erase(kartotek);
                  close(uj_kartotek);
                  rename(uj_kartotek,kartoteknev);
              end
  end;

begin { F O P R O G R A M }
  ismertetes;
  keszlet;
  valtoztatott:=false;
  tovabb:=false;
  repeat  {addig, amig tovabb=true }
      menu;
      case valasztas of
         'b','B':{ adattetelt bemutatni}
                 begin
                    assign(kartotek,kartoteknev);reset(kartotek);
                    tetelszam:=filesize(kartotek);
                    repeat  { amig 0-t nem irunk be }
                        clrscr;writeln;
                        tetelbemutatas('b');
                        if lapok_nr>0 then
                            begin
                               writeln;
                               write('Ker meg egy tetelt?...(I/N)');
                               readln(beiras);
                            end
                    until (lapok_nr=0) or not (beiras in ['i','I']);
                    close(kartotek);
                 end;
         'h','H':{ adatokat beirni }
                 begin
                    assign(kartotek,kartoteknev);reset(kartotek);
                    seek(kartotek,filesize(kartotek));
                    repeat
                       iras;
                       writeln;
              write('Hozzategyek meg egy tetelt?...(I/N)');
              readln(beiras);
           until not (beiras in ['i','I']);
           close(kartotek);
        end;
't','T':{ adatok torlese }
        begin
           assign(kartotek,kartoteknev);reset(kartotek);
           tetelszam:=filesize(kartotek);
           repeat { ismetelni, amig 0-t irunk be }
              clrscr; writeln;
              tetelbemutatas ('t');
              if lapok_nr >0 then
                            begin
                               writeln;
                               write('Igazan torolni akar?..(I/N)');
                               readln(beiras);writeln;
                               if beiras in ['i','I'] then
                                  begin
                                    torles(lapok_nr);
                                    writeln(' adattetelt toroltem!');
                                    writeln;
                                  end;
                               writeln;writeln;
                               write('Toroljek meg egy tetelt?..(I/N)');
                               readln(beiras);
                            end
                    until (lapok_nr=0) or not (beiras in ['i','I']);
                    close(kartotek);
                 end;
         'v','V':{Program Vege!}
                 begin
                    vege;
                    tovabb:=true
                 end;
      end
  until tovabb;
end.
```

Új fogalmak a Pascalban:

Állományok kezelése (FILE OF, Assign, Rewrite, Reset, Seek, Close, Eof, Filesize), új adattípusok definiálása RECORD-dal

FÜGGELÉK

A. ITERÁCIÓ ÉS REKURZIÓ A BASIC-BEN, A LOGO-BAN ÉS A PASCALBAN

A matematikában lehetőség van egy számérték *rekurzív* definiálására. Ennek klasszikus példája a *Fibonacci* néven is ismert *Leonardo de Pisa* nyúl feladata. A feladat:

„Hány nyúlpárunk lesz 3, 4, 5, ..., n hónap múlva, ha egy nyúlpár kéthónapos kortól kezdve havonta egy-egy új párt hoz világra, az új párok ugyanilyen törvény szerint szaporodnak és mind életben maradnak?"

Ezekkel a feltételekkel a *Fibonacci-számok* sorát kapjuk: 1, 1, 2, 3, 5, 8, 13, 21, ..., k. Ennek a sornak minden eleme a két előző elem összege. Egy tetszőleges elemet tehát így írhatunk fel:

$$A_n = A_{n-1} + A_{n-2}$$

A BASIC-ben A_n-t a következő programmal számíthatjuk ki:

```
10 PRINT"A FIBONACCI-SOR HANYADIK ELEMET"
20 INPUT"SZAMOLJAM KI ";N
30 A=1:B=1:N=N-1
40 FOR I=1 TO N
50 C=A+B
60 B=A:A=C
70 NEXT I
80 PRINT B

READY.
```

Ezt a programot közvetlenül átírhatjuk LOGO-ba:

```
TO FSTART1
 PRINT1 [HANYADIK FIBONACCI SZAMOT SZAMOLJAM KI?] CR 1
 MAKE "N INPUT
 FIBO :N 1 1
END

TO FIBO :N :A :B
 IF :N = 1 THEN PRINT :B TOPLEVEL
 MAKE "C :A + :B
 MAKE "B :A
 MAKE "A :C
 FIBO ( :N - 1 ) :A :B
END
```

A FSTART1 kezdő eljárással hívjuk meg a számítási eljárást, úgy, hogy a képernyőn bekért, keresett Fibonacci-szám sorszámát paraméterként átadjuk. Ezt a megoldást nevezzük *iteratív* megoldásnak.

Másfajta megoldás — amely a BASIC-ben nem lehetséges —, a LOGO-ban a feladat *rekurzív* megoldása. A rekurzió azt jelenti, hogy egy eljárás önmagát hívja meg.
A program indítását az előzőtől alig különböző FSTART2 eljárás végzi:

```
TO FSTART2
 PRINT1 [HANYADIK FIBONACCI SZAMOT SZAMOLJAM KI?] CR 1
 MAKE "N INPUT
 PRINT FI :N
END
```

Itt az **:N**-nel ismét a keresett Fibonacci-szám sorszámát adjuk át a FI számítási eljárásnak.

```
TO FI :N
 IF :N = 1 THEN OP 1
 IF :N = 2 THEN OP 1
 OP ( FI :N - 1 ) + ( FI :N - 2 )
END
```

Esetünkben a megoldást *kétszeres rekurziónak* nevezzük, mivel az eljárásban az eljárás kétszer hívja önmagát.
A rekurzív megoldások előnyét hívei a program eleganciájában látják: a FI először sorrendben megadja a két kezdeti értéket; majd szinte a matematikai megfogalmazással azonosan felírja a sorképzés törvényét. Kevesebb fáradságba kerül a programnyelv sajátságaihoz hozzáigazítani, mint a FIBO-t vagy a megfelelő BASIC megoldást.
A rekurzív megoldások hátránya a gyakorlatban mutatkozik meg. A felhasználónak, pl. a C 64-es LOGO-nál, már a 15. Fibonacci-számra is csaknem egy percet kell várnia, és 25.-re pedig már elfogadhatatlanul hosszú ideig. Ezzel szemben a számítási idő az iteratív megoldásnál a 15. számra alig észrevehető, és a 25.-nél is egy másodpercen belül van.
Azt, hogy mennyire különböző a két program tárigénye, úgy vizsgálhatja meg a legjobban, hogy begépeli a TRACE parancsot és lépésenként végezteti el a számítást. Így egyúttal azt is megfigyelheti, hogyan folyik a rekurzió a gépben. A közbenső hívások eredményeit a gép egymás után helyezi el, és amikor teljesül a leállási feltétel, ismét kiadja ezeket az eredményeket fordított sorrendben. Vagyis az utoljára kiszámított számot elsőként adja vissza a gép. A kívánt érték, az output (OP), a végén áll.
E könyvnek a keretei között nem áll módunkban a rekurzív megoldások előnyeit és hátrányait részletesen ismertetni. Ezért megelégszünk annyival, hogy a Pascal megoldáson belül egymás mellé állítjuk az *iteratív* és a *rekurzív* megoldást.

```
program fibonacci;
var szam            :integer;
    eredmeny        :real;
function frekur (x:integer):real;
  begin
     if x<=1 then frekur:=x
       else frekur:=frekur(x-1)+frekur(x-2)
  end;
function fiter (x:integer):real;
  var f0,f1,f2:real;
  begin
     f0:=0;f1:=1;f2:=1;
     while x>0 do
       begin
          f2:=f1+f0;
          f0:=f1;f1:=f2;
          x:=x-1
       end;
     fiter:=f0
  end;
begin  { foprogram }
  clrscr;
  writeln(' A Fibonacci-sor hanyadik elemere kivancsi?');
  readln(szam);writeln;
  write('rekurziv ----------> ');
  writeln(frekur(szam):1:0);writeln;
  write('iteracio ----------> ');
  writeln(fiter(szam):1:0)
end.
```

Itt is feltűnhet, hogy a rekurzív megoldás rövidebb és áttekinthetőbb, mint az iteratív, de a számítási idő itt is több (hosszabb) a rekurzív megoldásnál, mint az iteratívnál. Ha nem a számítási idő a fontos, hanem a feladat számítógépre való alkalmazása áll az előtérben, a rekurzív megoldás az előnyösebb.

B. LOGO—LAYOUT

A LOGO a sajátságaiból adódóan kevés lehetőséget kínál áttekinthető képernyő kialakítására. Szerencsére a felhasználó kibővítheti a LOGO nyelvet, vagyis magunk is írhatunk eljárásokat, amelyeket mint LOGO parancsokat használhatunk a programozásban. Ezeket azután kiegészítésképpen betölthetjük a számítógépbe. Ezt a lehetőséget használjuk ki, hogy megkönnyítsük a képernyőszerkesztést. Megfelelő LOGO eljárásokat definiálunk, és egy LAYOUT nevű állományban helyezzük el azokat. Így ezek az eljárások

READ "LAYOUT

utasítással bármikor betölthetők a tárba az esetleg már rendelkezésre álló eljárások mellé. Ettől kezdve ezek a segédeljárások úgy működnek a programban, mint maguk a LOGO szavak.
A következő eljárásokat használtuk:

1. Tabulátor

```
TO TAB :N :T
 CURSOR ( :N - 1 ) .EXAMINE 214 PRINT1 :T
END
```

A TAB eljárással egy soron belül tetszőlegesen át lehet helyezni az írás kezdőpontját. Az eljárás két paramétert igényel. Az első egy szám, amely megadja a képernyőn az új írási pozíciót, a második pedig egy szó (vagy egy numerikus változó értéke, vagy mondat, vagy lista), amit a gép a megadott írási pozícióban kezdve kiír. Például: TAB 7 "Jozsef utasításra a képernyő 7-edik íráshelyétől kezdődően az aktuális sorba beírja a JOZSEF szót. Ellentétben a szokásos tabulátor funkciókkal a TAB-ot úgy definiáltuk, hogy tartalmazza a kiíratási parancsot is. Így a **:T** változóban van az a szöveg, amelyet a TAB előhívásakor kiíratunk. A TAB eljárás egyetlen egy utasítássorból áll.
Ebben a sorban használjuk a CURSOR LOGO szót. A CURSOR szintén két adatot igényel: oszlop és sorszámot. A CURSOR 7 5 tehát, azt jelenti, hogy a kurzor az 5. sor 7. oszlopába megy. A sorszám a különböző LOGO nyelvjárásokban nehézségeket okoz. Egyes LOGO változatok a CURSOR párjaként a BO kérdést használják, amellyel a kurzor helyzetét kapjuk meg. A BO-nak két outputja van; az első az oszlopszám, a második a sorszám. A LAST BO-val tehát a CURSOR-hoz szükséges második adatot kapjuk meg. A PRINT1 hatására a kurzor, közvetlenül az írás után, ugyanabban a sorban marad. (Szemben a PR (PRINT) írási paranccsal, amelynek a hatására a kurzor minden írási folyamat végén a következő sor elejére ugrik.)

A különféle számítógépeknél a feladatnak speciális megoldásai vannak. A C 64-esen például a következő TAB eljárást is használhatjuk:

```
TO TAB2 :N :T
 CURSOR ( :N - 1 ) LAST CURSORPOS PRINT1 :T
END
```

A .EXAMINE paranccsal bármelyik tárrekesz tartalmát kiolvashatjuk. A C—64-esnél a 214-es számú rekeszben a pillanatnyi kurzorhelyezthez tartozó oszlopszám található.

Az APPLE változat nem ismeri a BO-t. Itt a

```
TO TAB :N :T
 CURSOR .DEPOSIT 36 :N
END
```

eljárás vezet célhoz.

A többi számítógépnél a gépkönyvben kell utánanézni, hogy melyik tárrekeszben található a pillanatnyi kurzorhelyzet.

2. Adatbevitel

Az INPUT eljárás lehetővé teszi, hogy közvetlenül a billentyűzetről vigyünk be adatokat. Ez az eljárás azért kell, mert a különböző gépek itt is különböző LOGO szavakat használnak. A C 64-esre az eljárás a következő:

```
TO INPUT
 OUTPUT FIRST RQ
END
```

Az OP az OUTPUT rövidítése. Az OUTPUT az őt követő változó értékét az előhívó eljárásba viszi át. Ez az érték esetünkben a FIRST RQ. A FIRST valamilyen lista első elemét veszi. A listát RQ (REQUEST) állítja elő a billentyűzetről bevitt jelsorozatból.

A LOGO különbséget tesz szavak — amelyek jelekből állnak — és listák — amelyek szavakból vagy esetleg listákból állnak — között. Az, hogy egy változó értéke szó vagy lista, a feldolgozásnál derül ki.
Kérjük, hasonlítsa össze a következő parancsoknál az eredményt.
1. parancs:
MAKE ”L (WORD ”’[’ ”KULONBSEG ”’]’) PRINT :L
Eredmény: [KULONBSEG]
2. parancs:
MAKE ”L (WORD ”’[’ ”KULONBSEG ”’]’) PRINT FIRST :L
Eredmény: [
3. parancs:
MAKE ”L [[KULONBSEG]] PRINT :L
Eredmény: [KULONBSEG]

4. parancs:
MAKE ”L [[KULONBSEG]] PRINT FIRST :L
Eredmény: KULONBSEG

Az 1. parancsnál a kerek zárójelben álló WORD parancs a mögötte álló listát egy szóvá fűzi össze, tehát a KULONBSEG füzérhez kapcsolja jobbról és balról a szögletes zárójeleket. Az így kapott füzér lesz az **:L** változó értéke, tehát így alakul ki az eredmény.

A 2. parancsnál ugyanez a helyzet egészen a nyomtatási utasításig. A **PRINT FIRST :L** utasítás viszont az **:L** változó tartalmának csak az első jelét nyomtatja ki, mert a **FIRST :L** az első jelet levágja és csak ezt tartja meg. Így lesz az eredmény a [jel.

A 3. parancsnál **:L** változó értéke a szögletes zárójelbe tett kifejezés lesz, a nyomtatás eredménye tehát: [KULONBSEG].

A 4. parancsnál **:L** változó értéke szintén a [KULONBSEG] kifejezés lesz. a **PRINT FIRST :L** utasítás ezt a kifejezést egy listának tekinti, és ennek veszi az első elemét. Ez az első elem viszont maga a KULONBSEG szó, tehát ezt nyomtatja ki.

Mivel a LOGO az RQ utasítással a beírt jelsorozatból egy 1 elemű listát készít, és mi magával a jelsorozattal akarunk dolgozni az adott feladatban, a FIRST RQ-val alakítjuk vissza az 1 elemű listát jelsorozattá. Az RQ, a C 64-es LOGO beírási parancsa, más LOGO nyelvjárásokban más jelölést kaphat. Például az APPLE LOGO német változatában ennek a parancsnak a neve EINGABE.

3. Egyéb funkciók

A CR eljárással a paraméterében meghatározott számú soremelést hajthatunk végre:

```
TO CR :N
 REPEAT :N [PRINT "’’]
END
```

A CR 3 például három üres sort eredményez. A REPEAT az eljárásban az ismétlést jelenti, **:N** megadja az ismétlések számát. Az ismétlendő folyamat szögletes zárójelben áll. (PRINT1 utasítás utáni CR 3 csak két üres sort eredményez, mert az első soremelést elhasználjuk az első üres sorba lépéskor.)

Az U eljárás kényelmesebbé teszi a képernyő szerkesztését. Például U 7 ”= hatására 7 darab egyenlőségjel jelenik meg a képernyőn.

```
TO U :N :Z
 IF :N = 0 THEN STOP
 PRINT1 :Z
 U :N - 1 :Z
END
```

Ha **:N** nullával egyenlő, akkor visszatérünk a hívó eljárásba. Ha azonban **:N > 0**, akkor a gép **:Z** tartalmát kiírja, majd újból meghívja az U-t eggyel csökkentett **:N**-nel, egészen addig, amíg **:N** = **0** nem lesz.

Az utolsó eljárás fontos feladatot old meg. A LOGO az egyes szavak, listaelemek, parancsok stb. között elválasztójelként a szóközt használja. Ezért sajátosan kezeli az üres jeleket. Ha például Ön úgy akar kiíratni egy mondatot, hogy az egyes szavak egynél több szóközzel legyenek elválasztva, észre fogja venni, hogy a szokásos LOGO parancsokkal ez nem megy. A PRINT [EZ EGY   PÉLDA] parancs eredménye a képernyőn:

```
EZ EGY PÉLDA
```

A LOGO az EGY és a PÉLDA közötti szóközöket egy szóközzé vonta össze. Ezen az akadályon segít át minket az S eljárás:

```
TO S :N
 U :N "'  '
END
```

amely egy soron belül **:N** szóközt ír ki.
Ezután a

```
PRINT1 [EZ EGY] S 7 PRINT [PÉLDA]
```

parancssor a kívánt eredményre vezet:

```
EZ EGY       PÉLDA
```

Az S eljárás az U eljárást hívja és átadja neki a kiírandó jelek számát, valamint kiírandó jelként a szóközt. Ilyenkor nem elég egyszerűen egy szóközt begépelnünk. Ezt ugyanis a LOGO nem fogadná el. Beírhatjuk viszont aposztrofok (") közé, s ekkor ezeket a LOGO szabályos jelként értelmezi.

Az itt leírt segédeljárások helyett természetesen más, megfelelő segédeljárásokat is definiálhattunk volna. A LOGO bővíthetőségével ellensúlyozza azt a hátrányát, hogy hiányzik belőle néhány egyszerű parancs. Segédeljárások helyett beírhattuk volna a megfelelő parancssorozatokat a programokba, de ezzel csak a terjedelmük bővülne feleslegesen.

Készült: Egyetemi Nyomda — 87.6399 Budapest, 1988
Felelős vezető: Sümeghi Zoltán igazgató